conference

proceedings

MobiSys 2005

The Third International Conference on Mobile Systems, Applications, and Services

Seattle, WA, USA

June 6–8, 2005

Jointly sponsored by
ACM SIGMOBILE and
The USENIX Association,
in cooperation with **ACM SIGOPS**

For additional copies of these proceedings contact:

USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710 USA
Phone: 510 528 8649
FAX: 510 548 5738
office@usenix.org
http://www.usenix.org

The price is $40.
Outside the U.S.A. and Canada, please add
$20 per copy for postage (via air printed matter).

Past MobiSys Conferences

2nd International Conference on Mobile Systems, Applications, and Services
June 6–9, 2004, Boston, MA, USA

First International Conference on Mobile Systems, Applications, and Services
May 5–8, 2003, San Francisco, CA, USA

ISBN 1-931971-31-5

Proceedings of the Third International Conference on Mobile Systems, Applications, and Services (MobiSys 2005)

Jointly sponsored by The USENIX Association and ACM SIGMOBILE,
in cooperation with ACM SIGOPS

June 6–8, 2005
Seattle, WA, USA

Conference Organizers

Program Co-Chairs

David Kotz, *Dartmouth College*
Brian Noble, *University of Michigan*

Program Committee

Tarek Abdelzaher, *University of Virginia*
William Arbaugh, *University of Maryland*
Elizabeth Belding-Royer, *University of California, Santa Barbara*
Brian Bershad, *University of Washington*
Lawrence Brakmo, *DoCoMo USA Labs*
Mark Corner, *University of Massachusetts*
David Culler, *University of California, Berkeley*
Nigel Davies, *Lancaster University*
Leendert van Doorn, *IBM Research*
Maria Ebling, *IBM Research*
Anthony LaMarca, *Intel Research, Seattle*
Eyal de Lara, *University of Toronto*
Margaret Martonosi, *Princeton University*
Chandra Narayanaswami, *IBM Research*
Mahadev Satyanarayanan, *Carnegie Mellon University*
Doug Terry, *Microsoft Research*
Roy Want, *Intel Research*
Rich Wolski, *University of California, Santa Barbara*

General Chair

Kang G. Shin, *University of Michigan*

Steering Committee Chair

Victor Bahl, *Microsoft Research*

Workshops Chair

Apratim Purakayastha, *IBM Research*

Poster/Demo Session Chair

Tarek Abdelzaher, *University of Virginia*

Work-in-Progress Session Chair

Lawrence Brakmo, *DoCoMo USA Labs*

Publicity Chair

Robin Kravets, *University of Illinois*

The USENIX Association Staff

External Reviewers

Thomas V. Frauenhofer
John Linwood Griffin
Wei Hong
Jason Liu
Trevor Pering
M.T. Raghunath
Marcel Rosu
Reiner Sailer
Alex Varshavsky
Xioalan Zhang

MobiSys 2005: Third International Conference on Mobile Systems, Applications, and Services
Seattle, WA, USA

Monday, June 6, 2005

Applications on the Go
Session Chair: Mark Corner, University of Massachusetts

Shake 'em, but Don't Crack 'em
Session Chair: Roy Want, Intel Research

Mobile Services
Session Chair: Mahadev Satyanarayanan, Carnegie Mellon University

Tuesday, June 7, 2005

Speedy Wireless
Session Chair: Brian Bershad, University of Washington

Tuesday, June 7, 2005 *(continued)*

Operating Systems for Sensor Networks
Session Chair: Doug Terry, Microsoft Research

Location (Here)
Session Chair: Anthony LaMarca, Intel Research, Seattle

Wednesday, June 8, 2005

Location (There)
Session Chair: Nigel Davies, Lancaster University

More Power to You
Session Chair: Chandra Narayanaswami, IBM Research

Index of Authors

Message from the Program Co-Chairs

Welcome to MobiSys 2005: The Third International Conference on Mobile Systems, Applications, and Services. We are honored to serve as the Program Co-Chairs of this premier event for publishing research about mobile and wireless systems. We thank everybody who submitted their papers to MobiSys, demonstrating the extensive work going on in this area, and the Program Committee and our external reviewers, who have spent countless hours providing feedback and guidance to create the final program. You all made our jobs much easier than we could have imagined possible!

There were 124 paper abstracts submitted to the conference and 84 full papers. This represented a substantial drop in submissions. However, we also noticed a significant decrease in out-of-scope submissions, particularly those focused on lower-level networking issues that are much more appropriate for MobiCom. As a result, the overall pool of papers from which to select was comparable to that seen in prior years.

Each full paper was assigned to three Program Committee members for a first round of review. Papers with an overall recommendation of "weak reject" or better, plus papers with high variance among reviewers, were assigned up to three additional Program Committee members for a second round of shorter, skim reviews. There were 50 papers selected for this second round. In toto, the Program Committee filed nearly 400 reviews. The committee convened in Ann Arbor, Michigan, on a chilly winter's day to discuss the 50 second-round papers. We ultimately selected 20 of these papers for inclusion in the conference, resulting in an acceptance rate of just under 24%. Each accepted paper was then shepherded by a PC member before being submitted for inclusion in the proceedings. A Best Paper Award is presented at the opening session of the conference.

In addition to the technical sessions, we have two invited speakers: Rick Rashid of Microsoft Research, discussing the difficulties inherent in using ubiquitous computing and communications to help bridge the information divide; and Alfred Spector of IBM Research, outlining the challenges in building robust, secure, and easy-to-administer pervasive systems. The conference also features a poster/demo session, along with work-in-progress reports—a busy few days!

The success of this conference is due entirely to the mobile systems community, the hard working teams that produce such excellent research, and the attendees, who create such a vibrant exchange of ideas. Welcome to MobiSys 2005.

David Kotz, Dartmouth College
Brian Noble, University of Michigan
Program Co-Chairs

A Systems Architecture for Ubiquitous Video

Neil J. McCurdy and William G. Griswold
Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093-0114
{nemccurd,wgg}@cs.ucsd.edu

Abstract

Realityflythrough is a telepresence/tele-reality system that works in the dynamic, uncalibrated environments typically associated with ubiquitous computing. By harnessing networked mobile video cameras, it allows a user to remotely and immersively explore a physical space. RealityFlythrough creates the illusion of complete live camera coverage in a physical environment. This paper describes the architecture of RealityFlythrough, and evaluates it along three dimensions: (1) its support of the abstractions for infinite camera coverage, (2) its scalability, and (3) its robustness to changing user requirements.

1 Introduction

Ubiquitous computing is often described as computers fading into the woodwork [18]. Ubiquitous video, then, is cameras fading into the woodwork, a notion captured by the expression, "the walls have eyes." Ubiquitous video is characterized by wireless networked video cameras located in every conceivable environment. The data is transmitted either to a central server or simply into the ether for all to view. Although many believe that such an environment is inevitable [2], we do not have to wait for the future to take advantage of ubiquitous video. There are a number of scenarios that could benefit from having live, situated access to ubiquitous video streams using today's technology.

Consider, for example, scenarios where it would be useful to attach head-mounted cameras to personnel entering dangerous, restricted, or remote sites. The video feeds can be streamed to a control "room" where commanders can navigate through the remote environment using the information acquired from the cameras. There are numerous such scenarios: In a disaster response setting, the failure to achieve adequate situational awareness can have catastrophic outcomes [13]. Live video situated in the disaster scene may be of benefit. Police Special Weapons and Tactics (SWAT) teams [6] that are routinely involved in high risk tactical situations may derive a similar benefit from live video. Other examples are: Hazardous Materials (HazMat) teams securing and decontaminating dangerous sites, police monitoring of events that attract large numbers of people such as holiday celebrations or protest marches, security personnel monitoring a remote site, and scientists studying a remote environment—one as benign as a nursery school or as dangerous as a volcano.

The common thread through this class of applications is that the harsh conditions of the real world need to be accomodated, and live, real-time access to the video is a requirement. Also, true, though, is that the accuracy of the data is far more critical than aesthetics. To help identify the minimum requirements of this class of applications, we will use a SWAT scenario as a specific example throughout this paper.

The key to harnessing ubiquitous video is in managing the incoming video streams. A naive approach would display the video on an array of monitors similar to those used in many building security systems today. An ideal solution would have infinite cameras in the field, and allow the user to move seamlessly through the environment choosing any desired vantage point. A more practical solution provides the illusion of the ideal system while operating under the constraints imposed by the real environment, including the constraint that the resulting displays should not be misleading.

We have created RealityFlythrough [11, 12], a system that uses video feeds obtained from mobile ubiquitous cameras to present the illusion of an environment that has infinite camera coverage. Stitching the multiple video streams together into a single scene is a straightforwardly sensible abstraction of numerous video streams. With such an abstraction, the user need only understand one integrated scene, as in a video game, rather than multiple feeds, as in a building security system. However, the limited number of cameras as well as the untamed elements of ubiquitous video make such an abstraction non-trivial to construct.

The key *limitation* of ubiquitous video is the incomplete coverage of the live video streams–every square meter of a space cannot be viewed from every angle with a live video stream at any chosen moment. For two cameras pointing in two rather different directions, when the user switches from viewing one camera to another, it

Figure 1: Snapshots of a transition. The transition uses two "filler" images to provide additional contextual information. During this transition the viewpoint moves roughly 20 meters to the right of the starting image and rotates 135 degrees to the right.

is often not obvious how the subject matter in the two views relate to each other, nor is it obvious what is in the intervening space between the two cameras.

To address this limitation, RealityFlythrough fills the intervening space between two cameras with older imagery (captured from the live camera feeds), and provides segues (i.e., transitions) between the two live cameras that sequences and blends the imagery in a way that provides the sensation of a human performing a walking camera pan. In certain scenarios the display of older imagery may be undesirable. While not ideal, transitions without background imagery are still sensible because the motion and timing of the transition and a background floor grid convey the distance and angle traveled. The user has complete control over how older imagery is displayed—whether it is displayed at all, in a sepia tone, or with an age-indicator-bar.

The key *untamed element* of ubiquitous video is the imprecision of the sensed location and orientation of a camera (due to both sensor latency and sensor inaccuracy). Such imprecision gives misleading cues to the user about how the subject matter seen in one camera relates to the subject matter in another. For example, the images might appear farther apart than they really are.

Under certain assumptions, offline vision techniques could perform seamless stitching [15]. To achieve real-time flythrough, this problem is instead handled by taking advantage of a property of the human visual system called *closure* [10]. *Closure* describes the brain's ability to fill in gaps when given incomplete information. It is a constant in our lives; *closure*, for example, conceals from us the blind spots that are present in all of our eyes. RealityFlythrough embraces *closure* by presenting the user with an approximate model of the relationships between two camera views, and having the user's visual cortex infer the relationships between the objects in the views. Dynamic transitions between still-images and live video feeds reveal the misregistrations in overlapping images (with an alpha blend), rather than hiding them through blending or clipping. Although this sacrificies aesthetics, the benefits obtained due to *closure* increase sensibility. For this technique to work, images must overlap. This property is sought by the mechanism that captures the older still-images for filling.

The contributions of this paper are the RealityFlythrough architecture, and its evaluation along three dimensions: (1) its support for the desired abstractions for ubiquitous video, (2) its scalability, and (3) its robustness to changing user requirements that is the measure of every good architecture.

The emphasis is on the architectural components that support the abstraction of infinite camera coverage. As will be shown throughout the paper, the architecture greatly reduces the complexity of the system, replacing complicated algorithms with concepts as simple as fitness functions. The design of a large-scale system that can accomodate thousands of cameras across multiple locations is considered in Section 7.3, but is not the focus of this paper. In many scenarios (most disaster response and SWAT scenarios), the size of the site and the availability of network bandwidth will limit the number of cameras that can be deployed. The architecture, as described, can easily handle these situations.

The architecture has two unique qualities. First, it uniformly represents all image sources and outputs as *Cameras*, supporting a rich yet simple set of operations over those elements in achieving the desired abstractions. And, second, it employs a separate *Transition Planner* to translate the user's navigation commands into a sensible sequence of camera transitions and accompanying image blends. Our experiments show good support for the desired abstractions, as well as excellent scalability in the number of live video sources and *Cameras*. Support for evolution is explored through a series of changes to the application.

The paper is organized as follows. Section 2 describes the user experience, and Section 3 compares our system to related work. Section 4 outlines the requirements of the system. We present a high level architectural overview of the system in Section 5, and then drill into the RealityFlythrough engine in Section 6 to reveal how the illusion of infinite cameras is achieved. Sections 7.1 and 7.2 evaluate the architecture's support of the system requirements, and Section 7.3 evaluates the architecture's tolerance to change and support for future enhancements. Section 8 concludes the paper.

2 User Experience

A significant part of the user experience in RealityFlythrough is dynamic and does not translate well to the written word or still-photographs. We encourage the reader to watch a short video [11] that presents an earlier version of RealityFlythrough, but we do our best to convey the subtlety of the experience in this section. When observing the images in Fig. 1, keep in mind that the transformation between the images is occurring within about one second, and the transitional frames represent only about 1/10th of the transition sequence.

The user's display is typically filled with either an image or a video stream taken directly from a camera. When the user is "hitchhiking" on a camera in this way, the experience is similar to watching a home-video where the camera operator is walking around while filming. A still-image, then, is simply the home-video paused. When a new vantage point is desired, a short transition sequence is displayed that helps the user correlate objects in the source image stream with objects in the destination image stream. These transitions are shown in a first person view and provide the users with the sensation that they are walking from one location to another. The illusion is imperfect, but the result is sensible and natural enough that it provides the necessary contextual information without requiring much conscious thought from the users.

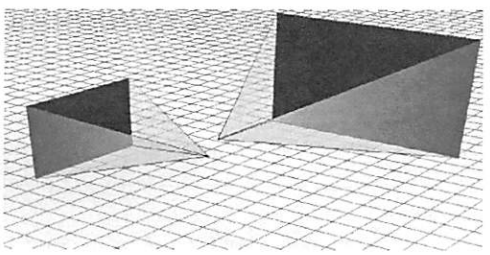


Figure 2: An illustration of how the virtual cameras project their images onto a wall.

RealityFlythrough works by situating 2d images in 3d space. Because the position and orientation of every camera is known, a representation of the camera can be placed at the corresponding position and orientation in virtual space. The camera's image is then projected onto a virtual wall (see Fig. 2). When the user is looking at the image of a particular camera, the user's position and direction of view in virtual space is identical to the position and direction of the camera. As a result, the entire screen is filled with the image. Referring to Fig. 1, a *transition* between camera A (the left-most image) and camera B (the right-most image) is achieved by smoothly moving the user's position and view from camera A to camera B while still projecting their images in perspective onto the corresponding virtual walls. By using OpenGL's standard perspective projection matrix to render the images during the transition, the rendered view situates the images with respect to each other and the viewer's position in the environment. Overlapping portions of the images are blended using an alpha-blend. By the end of the transition, the user's position and direction of view are the same as camera B's, and camera B's image fills the screen. As shown in Fig. 1, additional images are displayed (if available and if desired) to help provide contextual information.

It may be easier to understand how RealityFlythrough works by envisioning the following concrete example. Imagine standing in an empty room that has a different photograph projected onto each of its walls. Each image covers an entire wall. The four photographs are of a 360 degree landscape with one photo taken every 90 degrees. Position yourself in the center of the room looking squarely at one of the walls. As you slowly rotate to the left your gaze will shift from one wall to the other. The first image will appear to slide off to your right, and the second image will move in from the left. Distortions and object misalignment will occur at the seam between the photos, but it will be clear that a rotation to the left occurred, and the images will be similar enough that sense can be made of the transition. RealityFlythrough operates in a much more forgiving environment: the virtual walls are not necessarily at right angles, and they do not all have to be the same distance away from the viewer.

RealityFlythrough works in the wild because there is little information the system requires about each camera, and no preprocessing is required to render the transitions. The position of the camera can be obtained from whatever locationing technology is desired (we use WAAS-enabled consumer GPS's for outdoor tests), and the tilt, roll, and yaw can be determined with a tilt sensor that has a magnetic compass (we use an AOSI EZ-Compass). Since the human visual cortex is responsible for finding correlations between images the primary requirement for positional accuracy is that there be sufficient image overlap. We have found that an accuracy of 6-9 meters is adequate in outdoor settings where there is a wide field of view. Much higher accuracy would be necessary indoors—room-level accuracy, at minimum. Orientation accuracy is much more important because a camera that has less than a 40 degree field of view (typical of most web cameras) cannot be off by many degrees before images do not overlap at all. Magnetic compasses have provided good results, but may have trouble in areas of high magnetism.

3 Related Work

There have been several approaches to telepresence with each operating under a different set of assumptions. Telepresence [8], tele-existence [16], tele-reality [7, 15],

virtual reality and tele-immersion [9] are all terms that describe similar concepts but have nuanced differences in meaning. Telepresence and tele-existence both generally describe a remote existence facilitated by some form of robotic device or vehicle. There is typically only one such device per user. Tele-reality constructs a model by analyzing the images acquired from multiple cameras, and attempts to synthesize photo-realistic novel views from locations that are not covered by those cameras. Virtual Reality is a term used to describe interaction with virtual objects. First-person-shooter games represent the most common form of virtual reality. Tele-immersion describes the ideal virtual reality experience; in its current form users are immersed in a CAVE [3] with head and hand tracking devices.

RealityFlythrough contains elements of both tele-reality and telepresence. It is like telepresence in that the primary view is through a real video camera, and it is like tele-reality in that it combines multiple video feeds to construct a more complete view of the environment. RealityFlythrough is unlike telepresence in that the cameras are likely attached to people instead of robots, there are many more cameras, and the location and orientation of the cameras is not as easily controlled. It is unlike tele-reality in that the primary focus is not to create photo-realistic novel views, but to help users to internalize the spatial relationships between the views that are available.

All of this work (including RealityFlythrough) is differentiated by the assumptions that are made and the problems being solved. Telepresence assumes an environment where robots can maneuver, and has a specific benefit in environments that would typically be unreachable by humans (Mars, for example). Tele-reality assumes high density camera coverage, a lot of time to process the images, and extremely precise calibration of the equipment. The result is photorealism that is good enough for movie special effects ("The Matrix Revolutions" made ample use of this technology). An alternative tele-reality approach assumes a-priori acquisition of a model of the space [14], with the benefit of generating near photo-realistic live texturing of static structures. And finally, RealityFlythrough assumes mobile ubiquitous cameras of varying quality in an everyday environment. The resulting system supports such domains as SWAT team command and control support.

4 Requirements

In earlier work [12], we built a proof-of-concept system that revealed a number of rich requirements for harnessing ubiquitous video. Ubiquitous video is challenging because the cameras are everywhere, or at a minimum can go anywhere. They are inside, outside, carried by people, attached to cars, on city streets, and in parks. Ubiquity moves cameras from the quiet simplicity of the laboratory to the harsh reality of the wild. The wild is dynamic—with people and objects constantly on the move, and with uncontrolled lighting conditions; it is uncalibrated—with the locations of objects and cameras imprecisely measured; and it is variable—with video stream quality, and location accuracy varying by equipment being used, and the quantity of video streams varying by location and wireless coverage. Static surveillance-style cameras may be available, but it is more likely that cameras will be carried by people. Mobile cameras that tilt and sway with their operators present their own unique challenges. Not only may the position of the camera be inaccurately measured, but sampling latency can lead to additional errors.

Our proof of concept system revealed the need for better image quality, higher frame rates, greater sensor accuracy with faster update rates, and better support for the dynamic nature of ubiquitous video.

We used a SWAT team scenario as a concrete example to help us tease out the requirements of applications that may benefit from RealityFlythrough. At a typical SWAT scene, the team commander is situated some distance from the incident site, and often must direct field operations without the aid of visuals. Commands must be issued to field officers from their point of view, straining the commander's conceptual model of the scene. Mistakes do happen [6]. Discussions and initial trials with the San Diego Metropolitan Strike Team (MMST) with whom we are collaborating as a part of a larger project called WIISARD (Wireless Internet Information System for Medical Response) have confirmed that video may be an effective means for providing early situational awarenes. We can expect to have 25 officers, and therefore 25 cameras, in the field.

Common knowledge about police operations combined with the previous description reveal minimum requirements for a system that could support SWAT: The system must work at novel sites with minimal configuration; the command center must be nearby and fairly mobile; cameras should be mobile and therefore wireless; the system needs to handle very incomplete camera coverage with fewer than 25 cameras in the field; and the system must work in unforgiving environments with intermittent network connectivity.

5 System Overview

Given the requirements just outlined, how might such a system be built? First we need some cameras and location sensors. We need to capture the image data from a camera and compress it, and we also need to capture the sensor data. We call the components that do this, *Image Capture* and *Sensor Capture*, respectively. The data then needs to be combined so that we can match the sen-

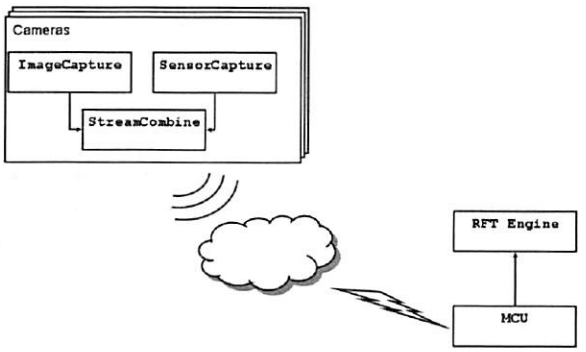


Figure 3: Component diagram showing system overview.

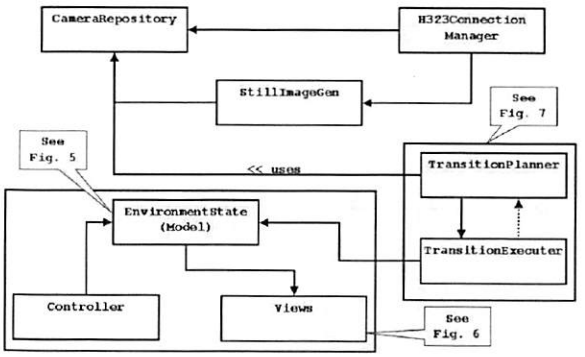


Figure 4: Component diagram showing an overview of the RealityFlythrough engine. Unlabeled arrows represent "calls" relationships. The dotted line is an event callback.

sor data to the appropriate frame in the image data. We call the component that handles this: *Stream Combine*. The resulting stream then needs to be sent across the network to a machine that decodes the data and presents it to the user. We have a modified *MCU* (Multipoint Control Unit) that does the decoding, and a *RealityFlythrough Engine* that combines the streams and presents the data to the user in a meaningful way. (Fig. 3 shows the relationships between these components.)

All of the video transmission components are based on the OpenH323 (`http://www.openh323.org`) implementation of the H323 video conferencing standard. Video can be transmitted using any H323 client without modification, but the sensor data would need to be transmitted separately and recombined on the server side. For our early prototype, though, we chose to embed the sensor data into the video stream to reduce complexity and to minimize network traffic. We mention this only because stand-alone video conferencing units that do hardware video compression are already starting to emerge, and it was a key design decision to follow standards so that we could support third party components.

RealityFlythrough is written in C++ and makes heavy use of OpenGL for 3D graphics rendering, and the boost library (`http://boost.org`) for portable thread constructs and smart pointers. A projection library (`http://remotesensing.org/proj`) is used to convert latitude/longitude coordinates to planar NAD83 coordinates, and the Spatial Index Library (`http://www.cs.ucr.edu/~marioh/spatialindex`) is used for its implementation of the R-Tree datastructure [5] that stores camera locations. RealityFlythrough is designed to be portable and is confirmed to work on both Windows and Linux.

The *Engine* is roughly 16,000 lines of code (including comments), and the *MCU* is an additional 2600 lines of code written on top of OpenH323.

6 Engine Architecture

The *RealityFlythrough Engine* is the heart of the system. Given the available video streams and the user's intentions as input, the *engine* is responsible for deciding which images to display at any point in time, and for displaying them in the correct perspective. Fig. 4 shows the functional components of the engine. The standard Model-View-Controller design pattern [4] is used to represent and display the current system state. The *Still Image Generator* is responsible for producing and managing the still-images that are generated from the live camera feeds. These still-images are used to backfill transitions, but may also be worth viewing in their own right since they may not be much older than the live feeds. The *Transition Planner/Executer* is responsible for determining the path that will be taken to the desired destination, and for choosing the images that will be displayed along that path. The *Transition Executer* part of the duo actually moves the user along the chosen path. And finally, the *Camera Repository* acts as the store for all known cameras. It maintains a spatial index of the cameras to support fast querying of cameras.

6.1 Model-View-Controller

The objects that comprise the Model-View-Controller support the abstraction of infinite camera coverage. The key element of our abstraction is a virtual camera (Fig. 5) which is simply a location, an orientation, a field of view, and a list of the "best" cameras that fill the field of view. The notion of "best" will be explored in Section 6.3, but for now simply think of it as the camera that most closely matches the user's wishes. A virtual camera, then, can be composed of multiple cameras, including additional virtual cameras. This recursive definition allows for arbitrary complexity in how the view is rendered, while maintaining the simplicity suggested by the abstraction: cameras with an infinite range of view exist at every conceivable location and orientation.

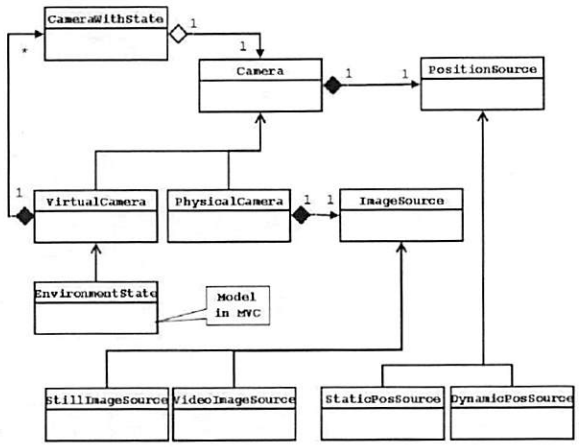


Figure 5: Class diagram showing the relationship of classes that are directly related to the Model in the MVC design pattern. For all class diagrams, open arrows represent inheritance, and arrows with diamonds represent containment. Open diamonds indicate that the container has only a reference to the object, while filled-in diamonds indicate ownership.

Model. The concept of a virtual camera is extended all the way down to the *Environment State* (Fig. 5) which is the actual *model* class of the Model-View-Controller. The user's current state is always using the abstraction of a *Virtual Camera* even if the user is hitchhiking on a *Physical Camera*. In that particular case the *Virtual Camera* happens to have the exact position, orientation, and field of view of a *Physical Camera*, and hence the physical camera is selected as the "best" camera representing the view. The current state of the system, then, is represented by a *Virtual Camera*, and therefore by a position, an orientation, and the physical cameras that comprise the view. Changing the state is simply a matter of changing one of these three data points.

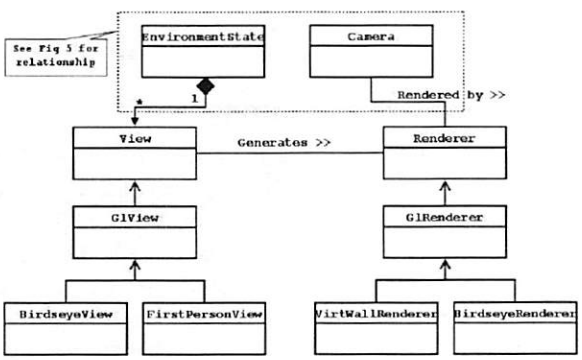


Figure 6: Class diagram for the classes involved in the View relationship of the MVC. The "Gl" in class names indicates that the classes are OpenGL-specific.

View The Model-View-Controller design pattern naturally supports multiple views into the system state. There are currently two views (Fig. 6), but we envision more (see Section 7.3). The two views are the *First Person View* and the *Birdseye View*. The *First Person View* is the primary view that displays the images from a first person immersive perspective. This is the view that was described in Section 2. The *Birdseye View* shows a top-down perspective on the scene, with cameras rendered as arrows and the field of view of active cameras displayed as cones emanating from the arrows (Fig. 7).

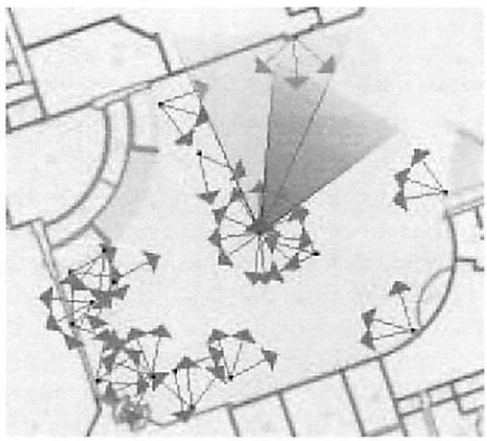


Figure 7: The birdseye view. The arrows represent the camera locations and directions of view.

The *Birdseye View* not only provides a wide-area map view of the scene, but also reveals some of the rawness of ubiquitous video that is being abstracted away by the *First Person View*. The birdseye view makes the live camera coverage (or lack thereof) obvious and it reveals the ages and density of the still-images that are used for backfill (see Section 6.2). There are currently three display modes available in the birdseye view: (1) show all cameras, (2) show only the cameras that have been updated within some user specifiable interval, and (3) show only the live cameras. In an ideal environment, the user could ignore the information presented in the birdseye view because a live image would be present at every vantage point. A more typical scenario, and the one we adopted in the experiment described in Section 7, presents the user with the birdseye view that shows only the locations of the live cameras. The assumption, then, is that the intervening space is fully populated with still-imagery. In this mode, the illusion of infinite camera coverage is still present, but the user is given some extra insight into where live camera coverage is available.

Each view instantiates one or more renderers to actually render the cameras that are involved in the current state. Since the definition of a *Virtual Camera* is recursive, there may be multiple cameras that need to be rendered. Each of these cameras has a state associated with it: the opacity (intensity) at which the camera's image

should be drawn for the alpha blend. There are currently two types of renderers: *Virtual Wall Renderer* and *Birdseye Renderer*.

The *Virtual Wall Renderer* is used by the *First Person View*. It renders images using the virtual wall approximation described in Section 2. The images are rendered in a specific order, on the appropriate virtual walls, and with the opacity specified in their state. The animation of a transition is achieved by moving the user's view point a set distance for each frame and progressing the alpha-blend for the overlapping portions of all of the images.

The *Birdseye Renderer* simply draws either the camera arrow or the frustum cone depending on the current state of the camera.

Controller The controller is a typical MVC controller and does not require further comment.

6.2 Still Image Generation

Key to the success of the infinite camera abstraction is the presence of sufficient cameras. If no imagery is available at a particular location, no amount of trickery can produce an image. To handle this problem, we take snapshots of the live video feeds and generate additional physical cameras from these. A *Physical Camera* consists of an *Image Source* and a *Position Source* (Fig. 5). The *Image Source* is a class responsible for connecting to an image source and caching the images. The *Position Source*, similarly, is responsible for connecting to a position source and caching the position. A camera that represents still-images, then, is simply a camera that has a static image source and a static position source. This is contrasted with live cameras that have a *Video Image Source* that continually updates the images to reflect the video feed that is being transmitted, and a *Dynamic Position Source* that is continually updated to reflect the current position and orientation of the camera.

To keep the still-imagery as fresh as possible, the images are updated whenever a camera pans over a similar location. Rather than just update the *Image Source* of an existing camera, we have chosen to destroy the existing camera and create a new one. This makes it possible to do a transitional blend between the old image and the newer image, without requiring additional programming logic. The images fit neatly into our *Camera* abstraction. We do not currently maintain a history of all still-images at a particular location, but it could be very useful to be able to move through time as well as space. We are moving to support this by saving video streams and allowing PVR-style (Personal Video Recorder) time-shifting.

The use of still-imagery to help achieve the abstraction of infinite camera coverage is of course imprecise. There are two ways that the limits of the abstractions are disclosed to the user:

First, the user has the option to never see older images. The user's preferences are used in the "best camera" calculation, and if no camera meets the criteria, the virtual camera will simply show a virtual floor grid.

Second, older images look different. The user can choose to have the old images displayed in a sepia tone, and can also choose whether or not to display an age-indicator-bar at the bottom of the sepia-toned or true-color images. The sepia tone makes it absolutely clear that the image is old, but it has the disadvantage that it alters the image, contradicting our aim to not mask reality. It is quite possible that this kind of image manipulation can hide information crucial to the user. An alternative is to show the age-indicator-bar on true-color images. The bar is bi-modal, giving the user high resolution age information for a short interval (we currently use 60 seconds), and lower resolution age information for a longer interval (currently 30 minutes). With a quick glance at the bottom of the screen, it is very easy for the user to get a sense of the age of an image.

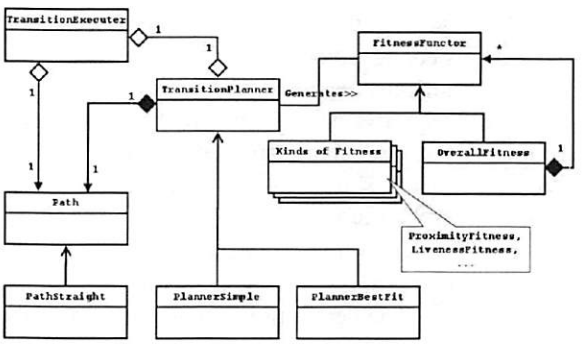


Figure 8: Class diagram showing the relationship of the classes involved in transition planning.

6.3 Transition Planner/Executer

When the user changes views, the *Transition Planner* (Fig. 8) is responsible for determining the path through space that will be taken and the images that will be shown along this path. The *Transition Executer* is responsible for moving the user along the chosen path. There is a high degree of coupling between the *planner* and the *executer* because of the dynamic nature of ubiquitous video. Consider a typical case where the user wishes to move to a live camera. A naive approach would determine the location and orientation of the live camera, compute the optimal trajectory to get to the target location and orientation, determine the images to be shown along the path, and finally execute the plan that was just developed.

This approach does not work in a ubiquitous video environment for several reasons. The primary problem

is that the destination camera may change its position and likely its orientation in the interval between when the plan was computed and when the execution of the plan has completed. The result will be a plan that takes the user to the wrong destination. Another problem is that the images that are selected along the path may not be the optimal ones. This is because the cameras that provide the intervening imagery may be live cameras as well, in which case their locations and orientations may have changed in the time since the plan was created. The result is that a live image that could have been shown is missed, or perhaps worse, a live image is shown that can no longer be seen from the current vantage point, so instead no image is displayed. Another possibility is that the dynamically generated still-imagery is updated after the plan is generated, but the older image is displayed instead.

To account for all of these problems the transition planning needs to be done dynamically and interleaved with the execution. There are a number of competing issues that need to be balanced when doing dynamic planning. It would seem that the ideal is to construct a plan at every time step, but some parts of the planning process are computationally expensive and need to be done sparingly. Also, the user needs to be given time to process the imagery that is being displayed, so even if a better image is available, showing it immediately may actually reduce comprehension.

The solution is to first introduce a dynamic *Path* object that takes a *Position Source* rather than a *Position* as its destination. The destination is now a moving target. At every time step, the *Path* can be queried to determine the current trajectory. With this trajectory, the *Transition Planner* can look ahead some interval and determine the best image to display. This image (camera, really) is added to the end of the camera queue. Each *Virtual Camera*—and since the *Transition Planner* acts on the *Environment State* remember that the *Environment State* is a virtual camera—maintains a fixed-length queue of cameras. When the queue is filled and a new camera is added, the camera at the front of the queue (the oldest or least relevant camera) is popped off the queue and thus removed from the *Virtual Camera*. The new camera that is added has a time-based opacity which means that the opacity gradually increases with time. We currently have the image blend to full opacity in one second.

This approach results in what appears to be a transition from one image to another, but along a dynamically changing path and with images that were used earlier still being displayed (if in view) to provide additional contextual information. The piece of the puzzle that is still missing is how the plan is constructed and adjusted dynamically. The *Transition Executer* (Fig. 8) is responsible for querying the *Path* at every time step and moving the user along the desired trajectory. It is also responsible for notifying the *Transition Planner* at time intervals set by the *planner*. These notification events give the *planner* the opportunity to determine which image (if any) to display next. Time is being used for signaling instead of "destination reached" because having the *Path* be dynamic means the destination may never be reached. Time is an adequate approximation of this signal point.

To determine the images to show during a transition the *Transition Planner* applies a series of *Fitness Functors* to each camera in the neighborhood. The *Fitness Functors* are weighted based on user preference. Some of the fitness dimensions are: proximity (how close is the camera to the specified position), rotation and pitch (how well do the orientations match), screen fill (how much of the screen would be filled with the image if it were displayed), recency (how recently was the image acquired), and liveness (is the camera live or not).

To further increase the sensibility of transitions, three heuristics are used to decide which images to display: (1) The current image should stay in view for as long as possible, (2) once the *to* image can be seen from the current position, no other images should be displayed, and (3) there should be a minimum duration for sub-transitions to avoid jumpiness. The first two items are handled by always applying the *Fitness Functors* to the current camera and the ultimate target camera regardless of whether they pass the "in the neighborhood test", and then boosting the fitnesses by a configurable scalar value. This has the effect of giving extra weight to the current and target cameras, thus indirectly satisfying our heuristics. The third item is handled by adjusting the time interval used for *Transition Planner* callbacks.

6.4 Camera Repository

The *CameraRepository* is simply a container for all of the cameras (including the still-cameras) that are known to the system. To support efficient spatial querying of the cameras, an R-Tree [5] is used to store the camera locations. The exact locations of the live cameras are not stored in the index because this would cause continuous updates to the index, and such precision is not necessary when doing "get cameras in neighborhood" queries. Instead, only location updates that are greater than a configurable threshold result in a replacement in the spatial index.

Each *physical camera* has certain fixed memory costs. To minimize the use of limited OpenGL resources, the cameras share a pool of texture maps. We have to store the image somewhere, though, so each camera (*Image Source*, really) allocates 768KB to store a 512x512 image (the size is dictated by OpenGL's texture map size requirements) at a depth of 24bits. After a period of inactivity, the *Image Source* frees memory by storing the

image to disk. Under normal loads, there is no perceptible difference in performance when an image is read from disk.

7 Evaluation

An architecture must be evaluated along two dimensions: does it work, and will it work in the future? In this section we first present a formative user study that captures the essence of the user experience and helps show that the abstractions presented are compelling and useful. Second, we examine performance to get insight into the scaleability of the system. Third, to evaluate how well the architecture will accommodate future changes to the application, we examine its robustness against a set of significant changes and extensions.

7.1 Effectiveness of the Abstraction

An earlier paper on RealityFlythrough [12] suggested that the first-person perspective used in transitions generated a sense of "being there". We re-ran this experiment using our new architecture which was designed to better handle the dynamic nature of a ubiquitous environment. Unlike the first experiment where the still-images were painstakingly pre-inserted, this run made full use of the automatic still-image capture described in Section 6.2. This user study and the one described in the earlier paper were formative studies designed to provide evidence that RealityFlythrough research is headed in the right direction.

To determine how the system was perceived by users, we repeated the earlier experiment as closely as possible. We used the same subjects, the same equipment on the user end, and the same location for the flythrough.

There were three hand-carried camera units in the field. They consisted of a standard logitech web camera (~$100), a WAAS-enabled Garmin eTrex GPS (~$125), a tilt sensor manufactured by AOSI (~$600), and an 802.11b equipped laptop. The tilt sensor provides compass, tilt, and roll readings at ~15hz. The video streams were transmitted using the OpenH323 video conferencing standard at CIF (352x288) resolution.

The subjects' task was to remotely explore our campus food court with the goal of getting a sense of what is happening, and to determine if there is anything to draw them to the site for lunch. The experiment was run twice because some problems with the system were encountered on the first run. We discuss this first experiment because the problems are revealing.

The first run of the new experiment was very positive from a technical standpoint. Three video streams connected successfully, and a large number of still-images were automatically generated, quickly filling the entire region with cameras. Only 61 pre-configured still-images were used in the earlier version of the experiment, but 100's were generated in this one, greatly increasing the camera density. Despite the extra overhead incurred by auto-generating the images and by planning transitions on the fly, the system performance felt about the same. In fact, the subjects made the statement that the "performance was definitely much nicer." The new H263 video codec proved to be far superior to the H261 codec used previously. The frame rate varied by scene complexity, but appeared to average about 6-8 frames per second. The frame size was the same as was used previously, but the image quality was better and the colors were much more vivid. The generated still-images were clear and of good quality. On several occasions the subjects rapidly pointed out the age of images, indicating the success of the age indicator bar.

Even with all of these improvements, though, the subjects were not left with a positive impression and had to conclude that "from a usability standpoint, it went down." Transition sequences were met with comments like "it seems like it's awkward to move through several of those stills", and "[that] transition wasn't smooth." Post-experiment analysis identified three sources for the problems: (1) Too many images were being presented to the user, not allowing time for one transition to be processed mentally before another one was started. (2) The attempt to acquire a moving target resulted in an erratic path to the destination, causing disorientation. And, (3) no attempt was made to filter the location data by sensor accuracy. Still-images were being generated even when the GPS accuracy was very low, so transitions involved nonsensical images which detracted from scene comprehension.

Fortunately, none of these problems were difficult to handle. In Section 7.3 we will discuss the actual modifications made because these unplanned changes exemplify the architecture's robustness to changing requirements.

The experiment was repeated with much more positive results. Despite worse conditions at the experiment venue (we shared the space with a well attended Halloween costume contest), the subjects had much more positive comments such as, "Let's try one in the completely opposite direction. That was pretty nice.", and "It's pretty accurate where it's placing the images." "That was kind of cool. They weren't quite all in the same line, but I knew and felt like I was going in the right direction."

The costume contest placed some restrictions on where the camera operators could go, forced them to be in constant motion, and resulted in a lot of close-range video footage of people's backs (the cameras were being held at chest level). The constant motion may be typical with head-mounted cameras, and should be adressed seriously. The subjects found the constant motion to be

annoying ("they're all over the map"), and the motion placed quite a strain on the new algorithm used to home in on a moving target. The subjects actually preferred the calmness of the still-images. Midway through the experiment, we asked the operators to slow down a bit, and the experience improved dramatically: "Yeah, that's what it is. So long as [the camera operators'] rotation is smooth and slow, you can catch up to it and have smooth transitions."

We have since experimented with ways to reduce the amount of motion that is experienced during transitions. The fact that our subjects preferred the calmness of the still-images is revealing. There were simply too many sources of movement in our transitions, making them difficult to comprehend and aesthetically unappealing. When we move through the real world we only have to take into account 6 dimensions of movement—our own movement in three dimensions and the movement of the objects we are viewing in three dimensions. During a transition involving a moving camera, however, the camera is moving independently and so represents another three dimensions that have to be processed. Each additional moving camera being displayed adds three more dimensions. There is simply too much movement to process. The solution we have adopted involves pausing the live video streams whenever they come into view during a transition, and playing them back at increased speed once they have been acquired. This approach will be described in more detail in Section 7.3.

7.2 System Performance

By measuring the performance of the system we hope to provide some insight into the scalability of the architecture. Raw performance metrics mainly measure the speed of the hardware and the quality of the compiler. Seeing how the raw numbers vary under certain conditions, however, reveals important details about the architecture.

The experiments with RealityFlythrough described thus far have only been run using at most three video streams. To determine the maximum number of simultaneous streams that can be handled by the server, we ran some simulations. The capacity of the wireless network forms the real limit, but since network bandwidth will continue to increase, it is instructive to determine the capacity of the server. We should estimate the capacity of a single 802.11b access point to give us a sense of scale, however. For the image size and quality used in the user studies, the H263 codec produces data at a relatively constant 200Kbps. Empirical study of 802.11b throughput has shown that 6.205Mbps is the maximum that can be expected for applications [17]. This same study shows that the total throughput drops drastically as more nodes are added to the system. With more than eight nodes, total throughput decreases to roughly 2Mbps. This reduction means we cannot expect to have more than 10 streams supported by a single 802.11b access point.

We will see that the bottleneck on the server is the CPU. As more compressed video streams are added to the system, more processor time is required to decode them. Some of the other functional elements in RealityFlythrough are affected by the quantity of all cameras (including stills), but the experimental results show that it is the decoding of live streams that places a hard limit on the number of live cameras that can be supported.

The machine used for this study was a Dell Precision 450N, with a 3.06Ghz Xeon processor, 512MB of RAM, and a 128MB nVidia QuadroFX 1000 graphics card. It was running Windows XP Professional SP2. The video streams used in the simulation were real streams that included embedded sensor data. The same stream was used for all connections, but the location data was adjusted for each one to make the camera paths unique. Because the locations were adjusted, still-image generation would mimic real circumstances. No image processing is performed by the engine, so replicating the same stream is acceptable for this study. The image streams were transmitted to the server across a 1 Gbit ethernet connection. Since the image stream was already compressed, very little CPU was required on the transmitting end. A 1 Gbit network can support more than 5000 simultaneous streams, far more than the server would be able to handle. Network bandwidth was not a concern.

To obtain a baseline for the number of streams that could be decoded by the server, we decoupled the *MCU* from the *engine*. In the resulting system, the streams were decoded but nothing was done with them. With this system, we found that each stream roughly equated to one percent of CPU utilization. 100 streams used just under 100 percent of the cpu. The addition of the 113th stream caused intermittent packet loss, with packet loss increasing dramatically as more streams were added. The loss of packets confirmed our expectation that the socket buffers would overflow under load.

Having confirmed that the addition of live cameras had a real impact on CPU utilization, we added the *RealityFlythrough engine* back to the system. We did not, however, add in the still-image generation logic. To determine the load on the system we looked at both the CPU utilization and the system frame rate as new connections were made. The system frame rate is independent of the frame rates of the individual video feeds; it is the frame rate of the transitions. It is desirable to maintain a constant system frame rate because it is used in conjunction with the speed of travel to give the user a consistent feel for how long it takes to move a certain distance. As with regular video, it is desirable to

have a higher frame rate so that motion appears smooth. To maintain a constant frame rate, the system sleeps for an interval between frames. It is important to have this idle time because other work (such as decoding video streams) needs to be done as well.

# Conn	~CPU (%)	~Fps achieved
10	85	15
15	**95**	**14**
20	100	10

Table 1: Observed behaviour when running the system at a system frame frate of 15fps. This table shows that the maximum number of simultaneous video streams that can be supported by our test hardware is 15. Adding more connections maxes out the CPU, making the system less responsive, and reducing the system frame rate.

For this experiment, we set the frame rate at 15fps, a rate that delivers relatively smooth transitions and gives the CPU ample time to do other required processing. As Table 1 indicates, fifteen simultaneous video feeds is about the maximum the system can handle. The average frame rate dips to 14fps at this point, but the CPU utilization is not yet at 100 percent. This means that occasionally the load causes the frame rate to be a little behind, but in general it is keeping up. Jumping to 20 simultaneous connections pins the CPU at 100 percent, and causes the frame rate to drop down to 10fps. Once the CPU is at 100 percent, performance feels slower to the user. It takes longer for the system to respond to commands, and there is a noticeable pause during the transitions each time the path plan is re-computed.

To evaluate the cost of increasing the number of cameras, still-image generation was turned on when the system load was reduced to the 15 connection sweet spot. Recall that still-images are generated in a separate thread, and there is a fixed-size queue that limits the number of images that are considered. Still-images are replaced with newer ones that are of better quality, and there can only be one camera in a certain radius and orientation range. What this means is that there are a finite number of still-images that can exist within a certain area even if there are multiple live cameras present. The only effect having multiple live cameras may have is to decrease the time it takes to arrive at maximum camera coverage, and to decrease the average age of the images. This assumes, of course, that the cameras are moving independently and all are equally likely to be at any point in the region being covered.

The live cameras were limited to a rectangular region that was 60x40 meters. A still-image camera controlled a region with a three meter radius for orientations that were within 15 degrees. If there was an existing camera that was within three meters of the new camera and it had an orientation that was within 15 degrees of the new camera's orientation, it would be deleted.

We let the system get to a steady state of about 550 still-images. The number of new images grows rapidly at first, but slows as the density increases and more of the new images just replace ones that already exist. It took roughly 5 minutes to increase from 525 stills to 550. At this steady state, we again measured the frame rate at 14fps and the CPU utilization at the same 95 percent. The system still felt responsive from a user perspective.

These results indicate that it is not the increase in cameras and the resulting load on the R-Tree that is responsible for system degradation; it is instead the increase in the number of live cameras, and the processor cycles required to decode their images. This shows that the architecture is scalable. Since the decoding of each video stream can be executed independently, the number of streams that can be handled should scale linearly with both the quantity and speed of the processors available. Depending on the requirements of the user, it is possible to reduce both the bandwidth consumed and the processor time spent decoding by throttling the frame rates of the cameras not being viewed. This would reduce the number of still-images that are generated; a tradeoff that only the user can make.

7.3 Robustness to Change

The investment made in an architecture is only warranted if it provides on-going value; in particular it should be durable with respect to changing user requirements, and aid the incorporation of the changes dictated by those new requirements. Below we discuss several such changes, some performed, others as yet planned. Only one of these changes was specifically anticipated in the design of the architecture.

7.3.1 Planned Modification

The hitchhiking metaphor has dominated our design up to this point. Another compelling modality for RealityFlythrough is best described as the virtual camera metaphor. Instead of selecting the video stream to view, the users choose the position in space that they wish to view, and the best available image for that location and orientation is displayed. "Best" can either refer to the quality of the fit or the recency of the image.

It should come as no surprise that the virtual camera metaphor inspired much of the present design, so there is a fairly straight-forward implementation to support it. The *Virtual Camera* is already a first class citizen in the architecture. To handle a stationary virtual camera, the only piece required is a *Transition Planner* that runs periodically to determine the "best" image to display. Part of the virtual camera metaphor, though, is supporting free motion throughout the space using video game style

navigation controls. The difficulty we will face implementing this mode is in minimizing the number of images that are displayed to prevent the disorienting image overload. This problem was easily managed with the hitchhiking mode because a fixed (or semi-fixed) path is being taken. The path allows the future to be predicted. The only predictive element available in the virtual camera mode is that the user will probably continue traveling in the same direction. It remains to be seen if this is an adequate model of behavior.

Another measure of a good architecture is that it is no more complicated than necessary; it does what it was designed to do and nothing more. The plan to support a virtual camera mode explains why the *Camera* is used as the primary representation for data in the system. Once still-images, video cameras, and "views" are abstracted as cameras, they all become interchangeable allowing for the simple representation of complicated dynamic transitions between images.

7.3.2 Unplanned Modifications

In Section 7.1 we described three modifications to the system that needed to be made between the first and seconds runs of the experiment. We also examine some recent changes to the system that address the user frustrations with there being too many sources of movement during transitions. Since all of these modifications were unplanned, they speak to the robustness of the architecture.

Reduce Image Overload. The goal of the first modification was to reduce the number of images that were displayed during transitions. This change had the most dramatic impact on the usability of the system, making the difference between a successful and unsuccessful experience. The modification was limited to the *Transition Planner*, and actually only involved tweaking some configuration parameters. In Section 6.3 it was revealed that the current and final destination cameras are given an additional boost in their fitness. Adjusting the value of this boost does not even require a re-start of the system.

In the time since our experiments were run, we have further improved the transitions by slightly modifying our approach to finding the next camera to display. Instead of looking ahead at a fixed time-interval, we now calculate when the current image will no longer be optimal—because it has rotated off-screen, it is zoomed in too near, or zoomed out too far—and use this time interval for selecting the next image. Each image is now displayed for an optimal amount of time. We still boost the fitness of the destination camera to reduce the number of images that are displayed as the transition nears completion. These changes were all confined to the *Transition Planner*.

Moving Camera Acquisition. The second modification also involved transition planning, but in this case the change occured in the *Path* class. The goal was to improve the users' experience as they transition to a moving target. The partial solution to this problem—implemented for the second experiment—adjusts the path that the users take so that they first move to the destination camera's original location and orientation, and then do a final transition to the new location and orientation. This makes the bulk of the transition smooth, but the system may still need to make some course corrections during the final transition. The full solution will be disclosed in the following section.

Too Much Movement. This modification has been made recently and has not been subjected to experimental evaluation. During the experiments our subjects voiced concern about the amount of movement experienced during transitions. Not only was the user virtually moving along the transition path, but the images generated by live cameras were also moving around the screen reflecting the camera movement as captured by the sensors. This problem was exacerbated by the seemingly erratic transition movement experienced during the acquisition of a moving target—helped, but not solved by the technique described in the previous paragraph.

Our current approach to this problem involves pausing the live video streams during a transition whenever they are visible on-screen. Once the destination camera has been acquired, the video stream is played back at increased speed until the users have caught up to the current time. This has two benefits: (1) it reduces the amount of motion that needs to be understood by the user during a transition, and (2) it pauses the moving target for an interval allowing for smoother final target acquisition. The video feeds are paused for only short durations (usually less than one second), so it does not take long for the user to catch up after the transition, and in early tests the pauses do not appear to be disruptive. The technique used for acquiring the moving target is still required because the target continues to move until it is actually visible on-screen.

Pausing of the video streams was handled by adding PVR-like (Personal Video Recorder) capabilities to the MCU. The incoming video streams are buffered to disk allowing for time-shifting and future replay of events. With this functionality added, the *Transition Planner* simply pauses and resumes the video feeds at the appropriate times.

Location Accuracy Filtering. The final change to the system was a little more substantial since it required modification to both the client and server software. The goal was to filter the still-images on location accuracy. This change would have been trivial if we were already

retrieving location accuracy from the sensors. As it was, the *Sensor Capture* component on the client had to be modified to grab the data, and then on the server side we had to add a location error field to our *Position* class. Now that every *Position* had an error associated with it, it was a simple matter to modify the *Still Image Generator* to do the filtering.

7.3.3 Future Modifications

Better High Level Abstraction. Forming continual correlations between the first-person-view and the 2d birdseye representation takes cognitive resources away from the flythrough scene and its transitions. We hope to be able to integrate most of the information that is present in the birdseye view into the main display. Techniques akin to Halos [1] may be of help.

This modification to the system should only affect the *First Person View*. Since we want to present the state information that is already available in the *Birdseye View*, that same information need only be re-rendered in a way that is consistent with the *First Person View*. If we want to create a wider field of view we could increase the field of view for the virtual camera that makes up the view. Another possibility is to generate additional views that are controlled by other virtual cameras. For example a window on the right of the display could be controlled by a virtual camera that has a position source offset by 45 degrees.

Sound. Sound is a great medium for providing context, and could be an inexpensive complement to video. By capturing the sound recorded by all nearby cameras, and projecting it into the appropriate speakers at the appropriate volumes to preserve spatial context, a user's sense of what is going on around the currently viewed camera should be enhanced.

Sound will be treated like video. Each *Physical Camera* will have a *Sound Source* added to it, and new views supporting sound will be created. There might be a *3D Sound View* which projects neighboring sounds, and a regular *Sound View* for playing the sound associated with the dominant camera.

Scale to Multiple Viewers with Multiple Servers. Currently RealityFlythrough only supports a single user. How might the system scale to support multiple users? The *MCU* component currently resides on the same machine as the *engine*. One possibility is to move the *MCU* to a separate server which can be done relatively easy since the coupling is weak. The problem with this approach, though, is that the *MCU* is decompressing the data. We would either have to re-compress the data, which takes time, or send the data uncompressed, which takes a tremendous amount of bandwidth. A better approach would be to leave the *MCU* where it is and introduce a new relay *MCU* on the new server layer. The purpose of the relay *MCU* would be to field incoming calls, notify the *MCU* of the new connections, and if the *MCU* subscribed to a stream, forward the compressed stream.

With the latter approach we could also support connecting to multiple servers. The *MCU* is already capable of handling multiple incoming connections, so the main issue would be one of discovery. How would the viewer know what server/s could be connected to? What would the topography of the network look like? We leave these questions for future work.

It is not clear where still-image generation would occur in such a model. The easiest solution is to leave it where it is: on the viewing machine. This has the additional benefit of putting control of image generation in the individual user's hands. This benefit has a drawback, though. Still images can only be generated if the user is subscribed to a particular location, and then only if there are live cameras in that location. What if a user wants to visit a location at night when it is dark? It's possible that the users want to see the scene at night, but it is equally likely that they want to see older daytime imagery. If the still-images are captured server side, this would be possible.

Since server-side still-image generation may stress the architecture as currently specified, we consider it here. The *engine* would not have to change much. We would need a *Still Image Generated* listener to receive notifications about newly generated cameras. A corresponding *Still Image Destroyed* listener may also be required. The camera that is created would have a new *Image Source* type called *Remote Image Source*. The *Position Source* would remain locally static. The *Remote Image Source* could either pre-cache the image, or request it on the fly as is currently done. Performance would dictate which route to take.

7.3.4 Robustness Summary

Each of the modifications presented is limited to very specific components in the architecture. This indicates that the criteria used for separating concerns and componentizing the system was sound.

8 Conclusion

We have presented an architecture for a system that harnesses ubiquitous video by providing the abstraction of infinite camera coverage in an environment that has few live cameras. We accomplished this abstraction by filling in the gaps in coverage with the most recent still-images that were captured during camera pans. The architecture is able to support this abstraction primarily because of the following design decisions:

(1) The *Camera* is the primary representation for data

in the system, and is the base class for live video cameras, still-images, virtual cameras, and even the environment state. Because all of these constructs are treated as a camera, they can be interchanged, providing the user with the best possible view from every vantage point.

(2) The *Transition Planner* is an independent unit that dynamically plans the route to a moving target and determines the imagery to display along the way. New imagery is displayed using an alpha blend which provides the illusion of seamlessness while at the same time revealing inconsistencies. The system provides full disclosure: helping the user make sense of the imagery, but revealing inconsistencies that may be important to scene comprehension. Because the *Transition Planner* is responsible for path planning, image selection, and the blending of the imagery, it has a large impact on the success of RealityFlythrough. Having the control of such important experience characteristics in a single componenent and having many of those characteristics be user controllable is key to the sucess of the current design.

The architectural choices made during the design of RealityFlythrough are primarily responsible for the effectiveness of the system. Complex algorithms that select the appropriate cameras to display at any given point are reduced to constructs as simple as fitness functions. The seemingly complicated rendering of multi-hop transitions to moving destinations is simplified to the rendering of a virtual camera from different perspectives along a dynamically changing path. The algorithms are simple; the architecture makes them so.

9 Acknowledgments

Special thanks to Robert Boyer, Jennifer Carlisle, Adriene Jenik, Charles Lucas, Michelle McCurdy, Jonathan Neddenriep, and the Active Campus team for their help with this project, and to our shepherd, Nigel Davies, whose comments on the paper were invaluable. This work was supported in part by a gift from Microsoft Research and contract N01-LM-3-3511 from the National Library of Medicine.

References

[1] P. Baudisch and R. Rosenholtz. Halo: a technique for visualizing off-screen objects. In *Proceedings of the conference on Human factors in computing systems*, pages 481–488. ACM Press, 2003.

[2] D. Brin. *The Transparent Society*. Perseus Books, 1998.

[3] C. Cruz-Neira, D. J. Sandin, and T. A. DeFanti. Surround-screen projection-based virtual reality: the design and implementation of the cave. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 135–142. ACM Press, 1993.

[4] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[5] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, pages 47–57. ACM Press, 1984.

[6] H. Jones and P. Hinds. Extreme work teams: using swat teams as a model for coordinating distributed robots. In *Proceedings of the 2002 ACM conference on Computer supported cooperative work*, pages 372–381. ACM Press, 2002.

[7] T. Kanade, P. Rander, S. Vedula, and H. Saito. Virtualized reality: digitizing a 3d time varying event as is and in real time. In *Mixed Reality, Merging Real and Virtual Worlds*. SpringerVerlag, 1999.

[8] H. Kuzuoka, G. Ishimo, Y. Nishimura, R. Suzuki, and K. Kondo. Can the gesturecam be a surrogate? In *ECSCW*, pages 179–194, 1995.

[9] J. Leigh, A. E. Johnson, T. A. DeFanti, and M. Brown. A review of tele-immersive applications in the cave research network. In *VR '99: Proceedings of the IEEE Virtual Reality*, page 180, Washington, DC, USA, 1999. IEEE Computer Society.

[10] S. McCloud. *Understanding comics: The invisble art*. Harper Collins Publishers, New York, 1993.

[11] N. J. McCurdy and W. G. Griswold. Tele-reality in the wild. UBICOMP'04 Adjunct Proceedings, 2004. `http://activecampus2.ucsd.edu/~nemccurd/tele_reality_wild_video.wmv`.

[12] N. J. McCurdy and W. G. Griswold. Harnessing mobile ubiquitous video. Technical Report CS2005-0814, University of California, San Diego, February 2005.

[13] McKinney and Company. *Post 9-11 Report of the Fire Department of New York*. August 2002.

[14] U. Neumann, S. You, J. Hu, B. Jiang, and J. Lee. Augmented virtual environments (ave): Dynamic fusion of imagery and 3d models. In *VR '03: Proceedings of the IEEE Virtual Reality 2003*, page 61. IEEE Computer Society, 2003.

[15] R. Szeliski. Image mosaicing for tele-reality applications. In *WACV94*, pages 44–53, 1994.

[16] S. Tachi. Real-time remote robotics - toward networked telexistence. In *IEEE Computer Graphics and Applications*, pages 6–9, 1998.

[17] A. Vasan and A. U. Shankar. An empirical characterization of instantaneous throughput in 802.11b wlans. `http://www.cs.umd.edu/~shankar/Papers/802-11b-profile-1.pdf`.

[18] M. Weiser. The computer for the 21st century. *Human-computer interaction: toward the year 2000*, pages 933–940, 1995.

LiveMail: Personalized Avatars for Mobile Entertainment

Miran Mosmondor
Ericsson Nikola Tesla, Krapinska 45, p.p. 93, HR-10 002 Zagreb
miran.mosmondor@ericsson.com

Tomislav Kosutic
KATE-KOM, Drvinje 109, HR 10 000 Zagreb
tomislav.kosutic@kate-kom.com

Igor S. Pandzic
Faculty of electrical engineering and computing, Zagreb University, Unska 3, HR-10 000 Zagreb
igor.pandzic@fer.hr

Abstract

LiveMail is a prototype system that allows mobile subscribers to communicate using personalized 3D face models created from images taken by their phone cameras. The user takes a snapshot of someone's face - a friend, famous person, themselves, even a pet - using the mobile phone's camera. After a quick manipulation on the phone, a 3D model of that face is created and can be animated simply by typing in some text. Speech and appropriate animation of the face are created automatically by speech synthesis. Animations can be sent to others as real 3D animated messages or as short videos in MMS. They can be used as fun messages, greeting cards etc. The system is based on a client/server communication model. The clients are mobile devices or web clients so messages can be created, sent and received on the mobile phone or on a web page. The client has a user interface that allows the user to input a facial image and place a simple mask on it to mark the main features. The client then sends this data to a server that builds a personalized face model. The client also provides an interface that lets the user request the creation of animated messages using speech synthesis. It is planned to have several versions of the client: Symbian, midlet-based, web-based, wap-based, etc. The server is responsible for sending messages adjusted to the capabilities of the receiving platform. The server, Symbian client, midlet-based client and the web client have been implemented as prototypes. We present the system architecture and the experience gained building LiveMail.

1. Introduction

Mobility and the Internet are the two most dynamic forces in communications technology today. In parallel with the fast worldwide growth of mobile subscriptions, the fixed Internet and its service offerings have grown at a rate far exceeding all expectations. The number of people connected to the Internet is continuing to increase and GPRS and WCDMA mobile networks are enabling connectivity virtually everywhere and at any time with any device. With advances in computer and networking technologies comes the challenge of offering new multimedia applications and end user services in heterogeneous environments for both developers and service providers.

The goal of the project was to explore the potential of existing face animation technology [11] for innovative and attractive services for the mobile market, exploiting in particular the advantages of technologies like MMS and GPRS. The new service will allow customers to take pictures of people using the mobile phone camera and obtain a personalized 3D face model of the person in the picture through a simple manipulation on the mobile phone. In this paper we present architecture of LiveMail system. We describe how unique personalized virtual characters are created with our face adaptation procedure. Also, we describe clients that are implemented on different platforms, most interestingly on mobile platform, since 3D graphics on mobile platforms is still in its early stages. Various different network and face animation techniques were connected into one complex system and we presented the main performance issues of such system. Also, the system uses a MPEG-4 FBA standard that could ultimately enable video communication at extremely low

bandwidths, and work presented in this paper could bring us one step further in that direction.

The paper is organized as follows. In the next section we give a brief introduction on used standards and technologies. Next, we present overall system architecture continuing with more details on server and client implementation in the following sections. Finally, system performance evaluation is given in the section 6.

2. Background

2.1. 3D modelling

Creating animated human faces using computer graphics techniques has been a popular research topic the last few decades [1], and such synthetic faces, or virtual humans, have recently reached a broader public through movies, computer games, and the world wide web. Current and future uses include a range of applications, such as human-computer interfaces, avatars, video communication, and virtual guides, salesmen, actors, and newsreaders [12].

There are various techniques that produce personalized 3D face models. One of them is to use 3D modeling tool such as 3D Studio Max or Maya. However, manual construction of 3D models using such tools is often expensive, time-consuming and it sometimes doesn't result with desirable model. Other way is to use specialized 3D scanners. In this way face models can be produced with very high quality using them in a mobile environment is not practical. Also, there have been methods that included use of two cameras placed at certain angle and algorithms for picture processing to create 3D model [8]. Some other methods, like in [9], use three perspective images taken from a different angles to adjust deformable contours on a generic head model.

Our approach in creating animatable personalized face models is based on face model adaptation of existing generic face model, similar to [14]. However, in order to achieve simplicity on a camera-equipped mobile device, our adaptation method uses a single picture as an input.

2.2. Face animation

Created personalized face model can be animated using speech synthesis [10] or audio analysis (lip synchronization)[13]. Our face animation system is based on the MPEG-4 standard on Face and Body Animation (FBA) [5][2]. This standard specifies a set of Facial Animation Parameters (FAPs) used to control the animation of a face model. The FAPs are based on the study of minimal facial actions and are closely related to muscle actions. They represent a complete set of basic facial actions, and therefore allow the representation of most natural facial expressions. The lips are particularly well defined and it is possible to precisely define the inner and outer lip contour. Exaggerated values permit to define actions that are normally not possible for humans, but could be desirable for cartoon-like characters.

All the parameters involving translational movement are expressed in terms of the Facial Animation Parameter Units (FAPU) (Figure 1.). These units are defined in order to allow interpretation of the FAPs on any facial model in a consistent way, producing reasonable results in terms of expression and speech pronunciation. They correspond to fractions of distances between some key facial features (e.g. eye distance). The fractional units used are chosen to allow enough precision.

Figure 1. A face model in its neutral state and defined FAP units (FAPU) (ISO/IEC IS 14496-2 Visual, 1999)

The FAP set contains two high level FAPs for selecting facial expressions and visemes, and 66 low level FAPs. The low level FAPs are expressed as movement of feature points in the face, and MPEG-4 defines 84 such points (Figure 2.). The feature points not affected by FAPs are used to control the static shape of the face. The viseme parameter allows rendering visemes on the face without having to express them in terms of other parameters or to enhance the result of other parameters, insuring the correct rendering of visemes. A viseme is a visual correlate to a phoneme. It specifies shape of a mouth and tongue for minimal sound unit of speech. In MPEG-4 there are 14 static visemes that are clearly distinguished and included in the standard set. For example, phonemes /p/, /b/ and /m/ represent one type of viseme. Important thing for their visualization is that the shape of the mouth of a speaking human is not only influenced by the current phoneme, but also by the previous and the next phoneme. In MPEG-4, transitions from one viseme to the next are

defined by blending only two visemes with a weighting factor. The expression parameter allows definition of high-level facial expressions.

The FAPs can be efficiently compressed and included in a Face and Body Animation (FBA) bitstream for low bitrate storage or transmission. An FBA bitstream can be decoded and interpreted by any MPEG-4 compliant face animation system [3][4], and a synthetic, animated face be visualized.

2.3. 3D graphics on mobile devices

Important aspect of our face animation system is 3D graphics on mobile phones. The last few years have seen dramatic improvements in how much computation and communication power can be packed into such a small device. Despite the big improvements, the mobile terminals are still clearly less capable than desktop computers in many ways. They run at a lower speed, the displays are smaller in size and have a lower resolution, there is less memory for running the programs and for storing them, and the battery only last for a short time.

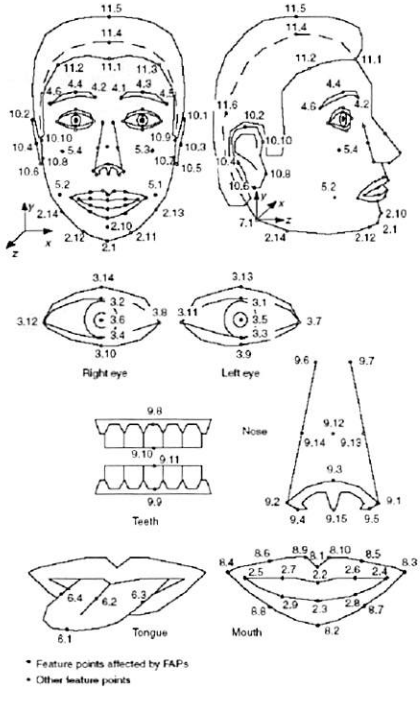


Figure 2. Face feature points (ISO/IEC IS 14496-2 Visual, 1999)

Rendering 3D graphics on handheld devices is still a very complex task, because of the vast computational power required to achieve a usable performance. With the introduction of color displays and more powerful processors, mobile phones are becoming capable of rendering 3D graphics at interactive frame rates. First attempts to implement 3D graphics accelerators on mobile phones have already been made. Mitsubishi Electric Corp. announced their first 3D graphics LSI core for mobile phones called Z3D in March 2003. Also other manufacturers like Fuetrek, Sanshin Electric, Imagination Technologies and ATI published their 3D hardware solution for mobile devices a few months after.

Beside hardware solutions, other important thing for 3D graphics on mobile devices is availability of open-standard, well-performing programming interfaces (APIs) that are supported by handset manufacturers, operators and developers alike. These are OpenGL ES (OpenGL for Embedded Systems) and Java Mobile 3D graphics (also known as JSR 184) that have emerged in last several months.

3. System architecture

The basic functions of the LiveMail service are simple creation of personalized face model, transmission and display of such virtual character on various platforms. The creation of such personalized face model consists of recognition of characteristic face lines on the taken picture and adjustment of generic face model to those face lines. The most important face lines are size and position of face, eyes, nose and mouth. Speech animation of the character is created from the input text. More details on the each module implementation are given in next chapters while this chapter describes system architecture.

Personalization of virtual characters and creation of animated messages with speech and lips synchronization is a time-consuming process that requires more computational power then the phone provides. Our system architecture is defined by this limitation. Capabilities of mobile devices have improved in last few years, but they are still clearly not capable of such demanding computing. Thus, our system is not implemented on one platform, rather is divided in several independent modules. The system is based on a client/server communication model. The basic task of the server is to perform computationally expensive processes like personalization of virtual character and creation of animated messages with speech and lips synchronization. The client side is responsible for displaying the animated message of the personalized virtual character and handling user requests for the creation of a new personalized virtual character and its animated message through proper user interface (Figure 3.). In this way LiveMail clients are implemented on

various platforms: Symbian-based mobile devices, mobile devices with Java support (Java 2 Micro Edition), WAP and Web interfaces.

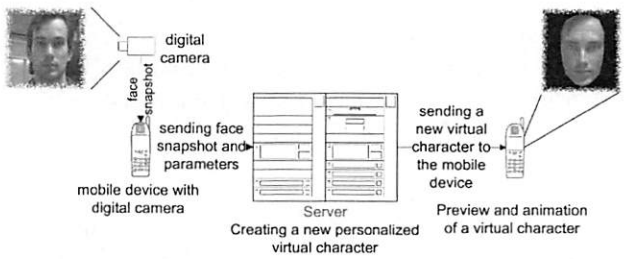


Figure 3. Simplified use-case scenario

The client application consists of a user interface through which users can create new personalized virtual characters and send animated messages, preview created messages and view received animated messages. When creating the personalized character the interface allows the user to input a facial image and place a simple mask on it to mark the main features. After selecting main face features the client sends this data to the server that builds a personalized face model (Figure 4.).

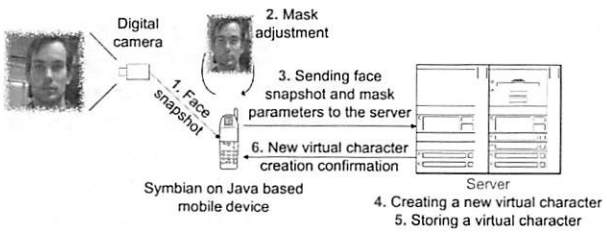


Figure 4. Personalized 3D face model creation

When creating animated messages, the user selects a virtual character, inputs text and addresses the receiver. The client application then sends a request for creation of the animated message to the server, which then synthesizes the speech and creates matching facial animation using the text-to-speech framework. The

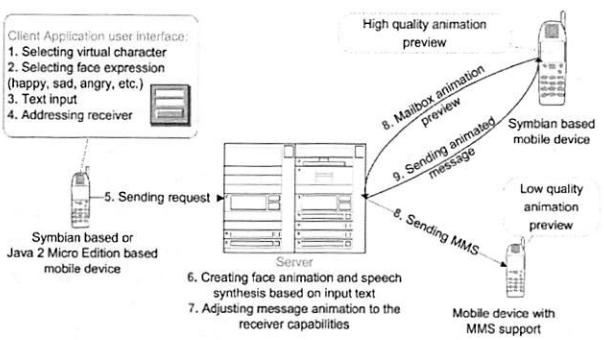


Figure 5. Creating animation message through client user interface

animated message is then adjusted to the capabilities of the receiving platform. For mobile devices that cannot display even medium quality 3D animation, animated message it is converted to a short video or animated GIF and sent by MMS (Figure 5.).

4. The server

The server is a combination of a light HTTP server and an application server (Figure 6.). The HTTP server provides clients with user interface and receives requests, while the application server processes client requests: creates new personalized 3D face models and animation messages. The user interface is dynamically generated using XSL transformations from an XML database each time client makes a request. The database holds information about user accounts, their 3D face models and contents of animated messages. The LiveMail server is a multithreaded application and the light HTTP server can simultaneously receive many client requests and pass them to the application server for processing. The application server consists of many modules assigned for specific tasks like: 3D face model personalization, animation message creation and more. During the process of 3D face model personalization and animated message creation there are resources that cannot be run in multithread environment. Therefore they need to be shared among modules. Microsoft's Text to speech engine, which is used for speech synthesis and as a base of 3D model animation, is a sample of a shared resource. To use such a resource all application server modules need to be synchronized.

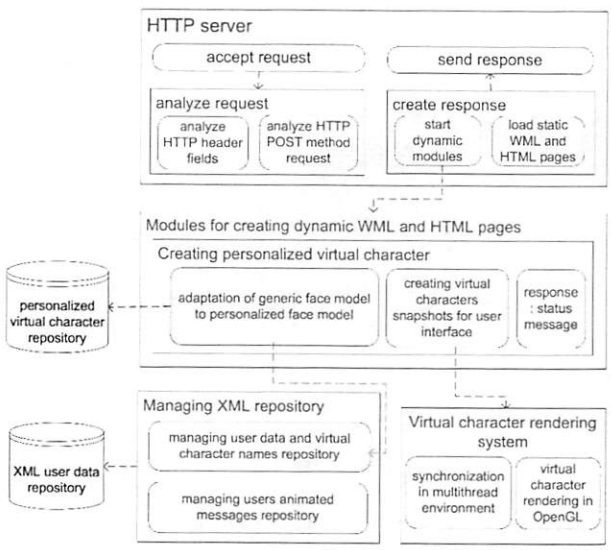


Figure 6. Simplified server architecture

The server's primary task is virtual character adaptation. The client's request for new virtual character creation holds entry parameters for adaptation process: the picture of person whose 3D model is created and the characteristic facial points in that picture (creating face mask). The server also has a generic 3D face model that is used in adaptation. Based on these inputs, the server deforms and textures the generic model in such a way that it becomes similar to the face in the picture, and thus produces the new 3D model ready for animation. The model is stored on the server for later use in VRML format.

The virtual character adaptation algorithm starts by mapping characteristic facial points from a facial picture to characteristic points in corresponding generic 3D face model. This initial mapping is followed by three main stages of adaptation:

The first stage is normalization. The highest and the lowest point of generic 3D face model are modulated according to the characteristic facial points from the facial picture. We translate the generic 3D face model to highest point from facial picture (point 11.4 on Figure 2.). The size of the translated model does not match the size of the facial picture mask so we have to scale it. We calculate vertical ratio of generic model and facial picture mask and move every point of the generic model in corresponding ratio. We distinguish vertical and horizontal scaling. In horizontal scaling we look at the horizontal distance between every point to highest point from facial picture, relative to face axis symmetry. Vertical scaling is easier, as there is no axis symmetry.

The second stage is processing. We distinguish texture processing and model processing. With N existing points we create a net of triangles that cover all normalized space. It's important to notice that beyond the face the net must be uniform. Based on known points and known triangles the interpolator algorithm is able to determine the coordinates of any new point in that space using interpolation in barrycentric coordinates. The interpolator used here is described in [15]. We forward characteristic points received from client's request to an interpolator. The interpolator (based on triangles net) will determine location of other points that we need to create new model.

The third stage of the algorithm is renormalization. It is practically the same algorithm as in first segment, except we roll back model from normalized space back to space where it was before starting of algorithm.

5. The client

The animation itself is created on the server that provides users with transparent access, meaning various client types could be used. After personalized face model with proper animation is created, it is send back to the client, where it can be previewed. Multi-platform delivery, and the capability to implement support for virtually any platform is one of the contributions of this system. Our strategy is to use a bare-minimum face animation player core. This core can be easily ported to any platform that supports 3D graphics.

The face animation player is essentially an MPEG-4 FBA decoder. When the MPEG-4 Face Animation Parameters (FAPs) are decoded, the player needs to apply them to a face model. Our choice for the facial animation method is interpolation from key positions, essentially the same as the morph target approach widely used in computer animation and the MPEG-4 Face Animation Tables (FAT) approach [2]. FATs define how a model is spatially deformed as a function of the amplitude of the FAPs. Interpolation was the earliest approach to facial animation and it has been used extensively. We prefer it to procedural approaches and the more complex muscle-based models because it is very simple to implement, and therefore easy to port to various platforms; it is modest in CPU time consumption; and the usage of key positions (morph targets) is close to the methodology used by computer animators and could be easily adopted.

The way the player works is the following. Each FAP (both low- and high-level) is defined as a key position of the face, or morph target. Each morph target is described by the relative position of each vertex with respect to its position in the neutral face, as well as the relative rotation and translation of each transform node in the scene graph of the face. The morph target is defined for a particular value of the FAP. The position of vertices and transforms for other values of the FAP are then interpolated from the neutral face and the morph target. This can easily be extended to include several morph targets for each FAP and use a piecewise linear interpolation function. However, current implementations show simple linear interpolation to be sufficient in all situations encountered so far. The vertex and transform movements of the low-level FAPs are added together to produce final facial animation frames. In case of high-level FAPs, the movements are blended by averaging, rather than added together.

Due to its simplicity and low requirements, the face animation player is easy to implement on a variety of

platforms using various programming languages. Additionally, for the clients that are not powerful enough to render 3D animations, the animations can be pre-rendered on the server and sent to the clients as MMS messages containing short videos or animated GIF images. In next chapters, we describe the following implementations of the client: the Symbian client, Java applet-based web client, and a generic mobile phone client built around J2ME, MMS and WAP. The first two implementations are full 3D clients, while the last one only supports pre-rendered messages.

5.1. Symbian client

Our mobile client is implemented on Symbian platform as a standalone C++ application. After taking a photo with camera, the user needs to adjust the mask with key face part outlined (Figure 7.). The mask is used to define 26 feature points on the face that are then, together with picture sent to the server for face adaptation, as described previously.

Figure 7. Symbian user interface and mask adjustment

After creation of personalized face model, it is sent back to the user where it can be previewed (Figure 8.). The face animation player on Symbian platform for mobile devices is based on DieselEngine. The DieselEngine is collection of C++ libraries that helps building applications with 3D content on various devices. DieselEngine has low-level API (Application Program Interface) that is similar to Microsoft DirectX and high level modules had to be implemented. The most important is a VRML parser that is used to convert 3D animatable face model from VRML format to Diesel3D scene format (DSC). Other modules enable interaction with face model like navigation, picking and centering.

Figure 8. 3D face model preview on Symbian platform

5.2. Java applet-based web client

A parallel web service is also offered as an interface to the system that allows users to access all the functionality: creating new personalized virtual characters and composing messages that can be sent as e-mail. The system keeps the database of all the created characters and sent messages for a specific user identified by e-mail and password.

As another way of creating new personalized virtual characters, the Java applet is used with the same functional interface described earlier. Compared to mobile input modules, the Java Applet provides more precision in defining feature points since the adjustment of the mask is handled in higher resolution of the desktop computer. Also the cost of data transfer, which is significantly cheaper compared to mobile devices, makes it possible to use higher resolution portrait pictures in making better looking characters.

Figure 9. Mask adjustment on Java applet-based web client

New messages can be made with any previously created characters. The generated animation is stored on the server and the URL to the LiveMail is sent to the specified e-mail address. The html page URL is produced dynamically. The player used to view LiveMails is a Java applet based on the Shout3D

rendering engine. Since it is a pure Java implementation and requires no plug-ins, the LiveMail can be viewed on any computer that has Java Virtual Machine installed.

5.3. Generic mobile client built around J2ME, MMS and WAP

The face animation player is easy to implement on a variety of platforms using various programming languages. However, for the clients that are not powerful enough to render 3D animations we an offer alternative: the animations can be pre-rendered on the server and sent to the clients as MMS messages containing short videos or animated GIF images.

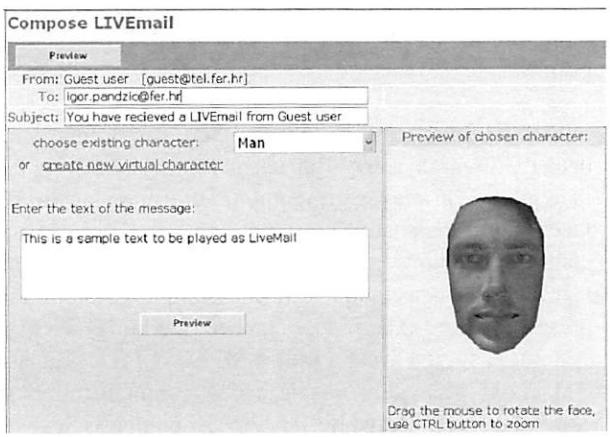


Figure 10. Composing LiveMail message on web client

A LiveMail WAP service is also provided for mobile phones. It provides the functionality of creating and sending LiveMail with previously generated personalized characters.

User interface through which personalized face model is created is also implemented on Java 2 Micro Edition (J2ME) platform. However, this platform alone does not define access to native multimedia services like, for example, camera manipulation and picture access. So, additional J2ME package, called Mobile Media API (MMAPI), was used.

6. System performance evaluation

In this section system performance evaluation is presented. First of all, the size of create new personalized virtual character request greatly depends on size of the face image. Other request data (user name, virtual character name, face mask parameters) is always approximately 1K. With average picture of 120x160 pixels request size is 15K. Request to create animated message carries few data and is always smaller than 1K.

6.1. Server

The described adaptation algorithm is very time consuming. Our measurements show that 98% of entire process is spent on the adaptation algorithm. The remaining 2% is spent on model storage and preview images for client user interface. With a generic 3D face model constructed of approx. 200 polygons adaptation algorithm takes 7.16 seconds on an AMD Athlon XP 2800+ (Figure 11.). Each adaptation algorithm started by client's request is run in a separate thread so multiprocessor systems can handle simultaneous client requests much faster.

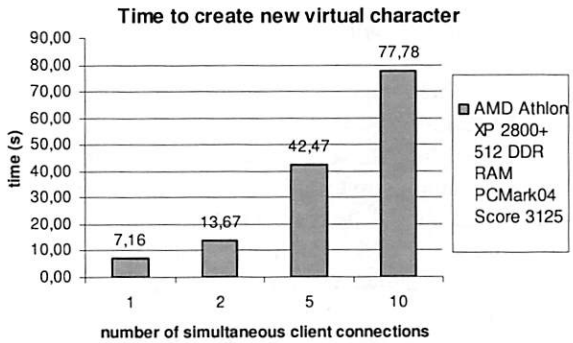


Figure 11. Time to create a new virtual character in respect to number of simultaneous client connections

The second most valuable server's task is creation of animated messages. Its execution time depends on message text length (Figure 12.). Also, Microsoft Text to Speech engine cannot process simultaneous text to speech conversions so all client requests are handled one at the time (while all others wait).

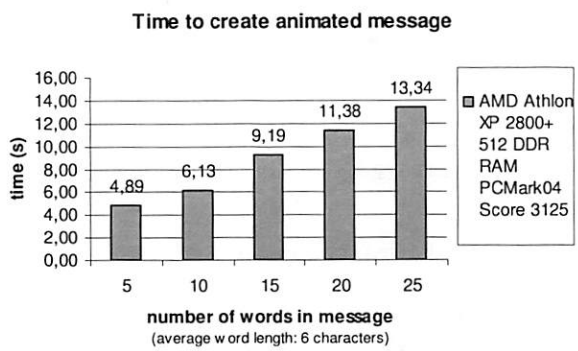


Figure 12. Time to create animated message in respect to number of words in message

6.2. Client

On the other side, required bandwidth for displaying animated message on Symbian client is approx 0.3 kbit/s

for face animation, and that is considerably less than sending raw video in any format. The reason for this is that FBA bitstream is built upon model based coding as was described previously. If we take analogy with voice, it uses the same principle like in GSM codec; it sends only parameters that are used on decoding side by a model to reproduce a sound, or in our case a face animation.

In addition to face animation, other bandwidth requirements for displaying animated message on Symbian client are 13 kbit/s for speech and approximately 50K download for an average face model. The Web client requires additional 150K download for the applet.

The Symbian client implementation was tested on Sony Ericsson P800 mobile device with various static face models. Interactive frame rates were achieved with models containing up to several hundreds polygons. The generic face model used in our LiveMail system uses approximately two hundred polygons and its performance is shown in Figure 13. After animation has started (in this case in 21st second), frame rate drops to average of 10 frames per second (FPS), but this is still relatively high above the considered bottom boundary for interactive frame rates on mobile devices.

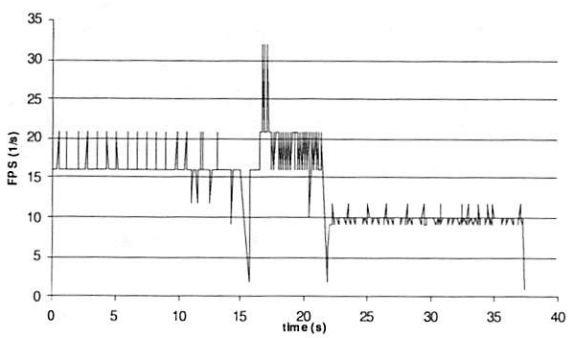


Figure 13. Generic face model performances on SE P800 mobile device

Web client showed performance of 24-60 fps with textured and non-textured face models of up to 3700 polygons on a PIII/1000. This performance is satisfactory for today's mobile PC user connecting to the Internet with, for example, GPRS. More details on this implementation and performance can be found in [11].

7. Conclusions

In this paper we have introduced a system that can be used as a pure entertainment application. Users deliver personalized, attractive content using simple manipulations on their phones. They create their own, fully personalized content and send it to other people. By engaging in a creative process - taking a picture, producing a 3D face from it, composing the message - the users have more fun, and the ways they use the application are only limited by their imagination.

LiveMail is expected to appeal to younger customer base and to promote services like GPRS and MMS. It is expected to directly boost revenues from these services by increasing their usage. Due to highly visual and innovative nature of the application, there is a considerable marketing potential. The 3D faces can be delivered throughout various marketing channels, including the web and TV, and used for branding purposes.

Besides entertainment aspects, there has been a great deal of research work. Barriers were crossed with the face animation player on mobile platform and with 3D face model personalization on the server. We have connected various different network technologies and face animation techniques in one complex system and presented the experiences gained building such a system. Also, the system uses a MPEG-4 FBA standard that could ultimately enable video communication at extremely low bandwidths. Although emphasis was on the entertainment, work presented in this paper could bring us one step closer to that goal.

8. Acknowledgments

We would like to acknowledge Visage Technologies AB, Linköping, Sweden, for providing the underlying face animation technology used for the system described in this paper.

References

[1] F.I. Parke, K. Waters, *Computer Facial animation*, A.K.Peters Ltd, 1996., ISBN 1-56881-014-8

[2] Igor S. Pandzic, Robert Forschheimer (editors), *MPEG-4 Facial Animation - The standard, implementations and applications*, John Wiley & Sons, 2002, ISBN 0-470-84465-5.

[3] M. Escher, I. S. Pandzic, N. Magnenat-Thalmann, *Facial Deformations for MPEG-4*, Proc. Computer Animation 98, Philadelphia, USA, pp. 138-145, IEEE Computer Society Press, 1998.

[4] F. Lavagetto, R. Pockaj, *The Facial Animation Engine: towards a high-level interface for the design of MPEG-4 compliant animated faces*, IEEE

Trans. on Circuits and Systems for Video Technology, Vol. 9, No. 2, March 1999.

[5] ISO/IEC 14496 - MPEG-4 International Standard, Moving Picture Experts Group, http://www.chiariglione.org/mpeg/standards/mpeg-4/mpeg-4.htm

[6] 3D Arts, DieselEngine SDK, http://www.3darts.fi/mobile/de.htm

[7] T. Fuchs, J. Haber, H.-P. Seidel, *MIMIC - A Language for Specifying Facial Animations*, Proceedings of WSCG 2004, 2-6 Feb 2004, pp. 71-78.

[8] Z. Liu, Z. Zhang, C. Jacobs, M. Cohen, *Rapid Modeling of Animated Faces From Video*, In Proceedings of The Third International Conference on Visual Computing (Visual 2000), pp 58-67, September 2000, Mexico City

[9] H. Gupta, A. Roy-Chowdhury, R. Chellappa, *Contour based 3D Face Modeling From A Monocular Video*, British Machine Vision Conference, 2004.

[10] Pelachaud, C., Badler, N., and Steedman, M., *Generating Facial Expressions for Speech*, Cognitive, Science, 20(1), pp.1–46, 1996.

[11] Igor S. Pandzic, Jörgen Ahlberg, Mariusz Wzorek, Piotr Rudol, Miran Mosmondor, *Faces Everywhere: Towards Ubiquitous Production and Delivery of Face Animation*, Proceedings of the 2nd International Conference on Mobile and Ubiquitous Multimedia, Norrkoping, Sweden, 2003

[12] Igor S. Pandzic, *Life on the Web*, Software Focus Journal, John Wiley & Sons, 2001, 2(2):52-59.

[13] P. Hong, Z. Wen, T. S. Huang, *Real-time speech driven Face Animation*, in I. S. Pandzic, R. Forchheimer, Editors, *MPEG-4 Facial Animation - The Standard, Implementation and Applications*, John Wiley & Sons Ltd, 2002.

[14] W.Lee, N.Magnenat-Thalmann, *Fast Head Modeling for Animation*, Journal Image and Vision Computing, Volume 18, Number 4, pp.355-364, Elsevier, March, 2000

[15] Igor S. Pandzic, *Facial Motion Cloning*, Graphical Models journal.

MediaAlert – A Broadcast Video Monitoring and Alerting System for Mobile Users

Bin Wei
Bernard Renger
Yih-Farn Chen
Rittwik Jana
Huale Huang
Lee Begeja
AT&T Labs-Research
180 Park Ave
Florham Park, NJ 07932
{bw, renger, chen,
rjana, huale, begeja}
@research.att.com

David Gibbon
Zhu Liu
Behzad Shahraray
AT&T Labs-Research
200 Laurel Ave S
Middletown, NJ 07748
{dcg, zliu, behzad}
@research.att.com

Abstract – We present a system for automatic monitoring and timely dissemination of multimedia information to a range of mobile information appliances based on each user's interest profile. Multimedia processing algorithms detect and isolate relevant video segments from over twenty television broadcast programs based on a collection of words and phrases specified by the user. Content repurposing techniques are then used to convert the information into a form that is suitable for delivery to the user's mobile devices. Alerts are sent using a number of application messaging and network access protocols including email, short message service (SMS), multimedia messaging service (MMS), voice, session initiation protocol (SIP), fax, and pager protocols. The system is evaluated with respect to performance and user experiences. The MediaAlert system provides an effective and low-cost solution for the timely generation of alerts containing personal, business, and security information.

Keywords: Mobile devices, multimedia processing, content repurposing, content adaptation, service platform, news monitoring, automatic speech recognition (ASR), multimedia messaging, alerting, notification.

1 INTRODUCTION

Mobile devices are gaining ground in computing power, storage capacity, and communication capabilities. A few years ago, Personal Digital Assistants (PDA) could only be used to store a limited amount of personal and other information. Today, these devices support standard peripherals such as digital cameras and the Global Positioning System (GPS), and have the ability to play video in stored or streaming modes. The processing power of mobile devices continues to advance according to Moore's law. For example, the clock rate of the Intel XScale PXA 27x processor is 624 MHz. The network bandwidth, wireless, storage, and graphic capabilities are growing at even faster rates. Compact flash memory cards with capacities as large as 12 GB are now available and capacities as high as 4 GB are affordable. Hard disks for certain mobile devices can accommodate over 60 GB of data. In addition, the latest smart phones such as the O2 Xda II can be equipped with both wireless LAN (WiFi) cards and GSM/GPRS capabilities. Mobile devices are now more than just digital organizers of yesterday and have evolved into powerful personal multimedia and communication devices. Consequently mobile devices are becoming so ubiquitous that they are literally part of the fabric of our lives and are so intuitive to use that we hardly notice them.

The storage capacity and multimedia presentation capabilities of mobile devices enable users to carry large collections of information with them. However, new information continues to become available on a daily basis. To gain access to such information in a timely manner, mobile users need to rely on the communications capabilities of these devices. Automated content processing and notification mechanisms are also needed to accommodate the capabilities of mobile devices and to be able to sift through large amounts of information.

We believe that the promise of mobile devices can be fully realized through a well-planned service architecture that can connect all relevant devices together and provide a common platform to develop and

deploy value-added services to meet time to market (TTM) and time to volume (TTV) requirements. We believe that this rapid evolution in mobile device technologies mandates a flexible service platform that can adapt information to new devices and access protocols easily.

TV broadcast news programs are a major source of information to TV viewers because they inform them what is happening today. In particular, mobile users want to be informed and in many cases they want to be informed as soon as new content is available. Thus, content that can be sent to mobile devices expeditiously would be desirable to mobile users. Our focus in this study is to automatically extract relevant video segments from broadcast news programs according to user's interests and to make the video content accessible to users through wired or wireless devices.

In this paper, we investigate how to build a service platform that would address the need for timely dissemination of information contained in broadcast news programs to mobile users. Our goal is to provide a common multimedia processing and alerting platform that enables innovative services to be deployed quickly. We believe that the novelty in this paper lies in our use of automated content processing techniques such as multimodal story segmentation combined with personalization both with respect to content selection and to content delivery options. This content preprocessing then enables the alerting platform to perform flexible content adaptation to provide users with the highest quality presentation given bandwidth and device capability constraints. The combined implementation of automated content processing and content adaptation results in a richer set of services for the end user than could be achieved if the separate system components were presented in isolation. Content adaptation and alert dissemination must be general purpose, handling any type of input and supporting a wide range of output devices. On the other hand, application-aware content processing can improve the user experience in restricted domains. For example, rather than just down-sampling video or extracting key frames, systems for delivering broadcast news content can leverage the closed caption text and filter the extracted key frames to produce a concise representation. This pre-processed content can then be further adapted for specific devices as necessary. In the rest of the paper, we first provide a general framework for media processing and alerting services. We then describe the architectural components for enabling such a system. The implementation section then describes system details and typical usage scenarios followed by a system evaluation from the perspective of performance and user experience. We give an overview of related work. After a brief discussion of future trends and some existing issues, we conclude the paper.

2 MEDIA ALERTING SERVICES FRAMEWORK

In this section, we consider a general framework for media alerting services and introduce the architectural components that are detailed in Section 3 for a broadcast video monitoring and alerting system.

2.1 A HIGH-LEVEL ABSTRACTION

There are three basic components in any alerting service: media acquisition, alert construction, and alert delivery. First, a mechanism to obtain media sources must be available. Media sources may be from public channels or private channels and may be presented in various forms, such as text, images, audio, video or any combination of these. Second, media processing mechanisms must be available to extract information from the media content. Third, alert information must be delivered to target users within a certain time period. The delivery mechanism depends on the devices accessible to end users.

Figure 1 presents a high-level logical framework of media alerting services. This includes the types of media sources that are used, how content is acquired and processed, and the target devices or protocols that are supported. The common goal of any media alerting system is to obtain relevant content segments from media sources and to deliver them automatically, regardless of where users are and what devices they are using.

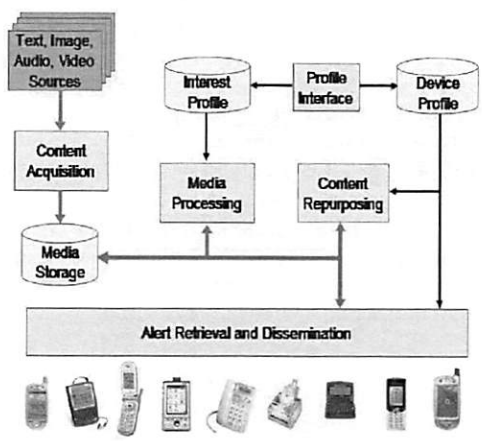


Figure 1 – General Framework for Media Alerting Services

The framework shown in Figure 1 consists of several components (a subset of them can be deployed for different implementations of media alert systems):

- Content Acquisition: The media sources can range from text, image, audio, to videos. Our focus is

primarily on video sources. Types of video feeds include terrestrial broadcast television, surveillance cameras, satellite broadcasts, IP streaming media. These feeds may vary widely in terms of content. In particular, the level of post-production processing in the media source has implications for media adaptation later on. Video processing techniques designed for unstructured video such as from a web camera or closed-circuit security feed may not be suitable for highly produced or structured television news material.

- Media Storage: After a piece of content is acquired, the media storage keeps both its initial raw format and successive transformations and adaptations for later use.
- Profile Interface: The profile interface collects user interests to help extract relevant media clips; it also collects device profiles and user configuration information that are then passed to the alert dissemination component.
- Media Processing: Media processing is used to detect relevant content, segment it and convert it into a form that is amenable for efficient processing later on. One common process employed is media segmentation, which is critical for alerting applications because long form content does not lend itself well to dissemination over messaging protocols such as SMS nor is it readily consumable on devices with limited user interface capabilities. By automatically segmenting the media based on topics, and adding this logical data structure to the multimedia database, we can rapidly produce smaller content units that are of a manageable size to satisfy bandwidth and device storage requirements.
- Content Repurposing: This process is needed to perform media adaptation to support a wide array of device types.
- Alert Dissemination: Given alert content and a list of recipient devices, this component is responsible for taking the repurposed content and delivering it via the appropriate access protocols. An important part of this module is handling the complexity of scheduling a very large number of alerts to meet stringent time constraints. This is a "Push" operation.
- Alert Retrieval: This component allows mobile users to query and retrieve alert content through different access protocols. This is a "Pull" operation.

Other components needed to produce a viable service that are not shown in Figure 1 include alert reporting/tracking and operations/systems support management functions.

2.2 Alert Types

In addition to the accuracy and correctness of media processing for extracting alert content, the delay is a factor that affects the overall user experience. It mainly depends on how the media processing is scheduled. From the user's perspective, there are several types of media alerts with respect to latency.

Scheduled Alerts: Users are alerted at specific times in the day of any alerts that may have occurred since the last scheduled alert time. This is suitable for users who prefer not to be alerted at odd hours but the users run the risk of not receiving alerts in a timely manner.

Immediate Alerts: In this case, the system runs in real-time and attempts to minimize the latency in delivering the alerts. It may attempt to deliver an alert as an event unfolds and as the media stream is still being acquired. For example, in the case of keyword spotting in broadcast video, it is a relatively easy task to build a system that has low latency from the time the last closed caption character of the keyword was broadcast to the time that the user gets notified. Topic spotting would be more difficult, e.g., looking for events such as earthquakes or corporate mergers. This is challenging to do while the text is streaming into the system. It also raises the possibility that the system might determine that the clip matches a user's interest profile before the clip finishes airing.

Predictive Alerts: In this case, the system has out of band information, such as an electronic program guide (EPG), which allows it to determine that content matching a user's profile will be available at some point in the future. Users would be alerted to "tune in" at the appropriate time. There are some well-known systems that use EPG and user interest profiles but they don't involve alerting on various mobile devices. Additional flexibility can be obtained if we don't require EPGs at all. We can analyze the content to determine if upcoming content will be of interest. For example: "Mt. St. Helens erupted and we will have a live video feed coming up shortly."

The system we will describe next provides the basis for handling all three types of alerts. However the application we developed is basically for *scheduled alerts.*

3 A Broadcast Video Monitoring and Alerting Service

We have built a media alerting service (called MediaAlert) that focuses on TV broadcast news as the

media source with the goal of delivering repurposed media alerts to a wide variety of mobile devices.

In this section, we describe the architectural components of the system based on the model presented in Section 2. MediaAlert is implemented by combining a media processing platform with a content delivery platform. The former is the eClips system [4], which is based on the Digital Video Library (DVL) platform [24]. The latter is the Alert Dissemination Engine built on top of the AT&T Enterprise Messaging Network[SM] platform [3]. This integration has several advantages:

- A large digital video library is available for users to search and retrieve video content from as early as the 1990s.
- Various formats of media content are derived from the original captured video including transcoded video, re-sampled audio streams, associated text either through closed captioning or ASR (Automatic Speech Recognition), and key frames. The derived content provides a rich set of resources for satisfying user's requests under various device constraints.
- The content delivery platform allows the use of different protocols to communicate with different devices. Thus the optimization of content repurposing can be achieved through the knowledge of the device profiles with the necessary transcoding from the appropriate media content.

The rest of this section will describe the system components of our media alerting service in detail.

3.1 Content Acquisition

MediaAlert currently records selected broadcast TV programs from several broadcasters using satellite or cable feeds based on a pre-determined schedule and according to the interests of the target audience. The structured video feeds from broadcast television are then digitized, compressed and stored in a multimedia database (see Media Storage in Figure 1).

The database also holds high level metadata relevant to the content feeds including electronic program guide (EPG) information such as program title, air date, broadcaster, etc. The content acquisition subsystem could take the form of a bank of digital video recorders linked to a centralized content store.

The EPG data is too sparse to provide focused, concise, multimedia information that is relevant to the users. To address this, we automatically process the content of media streams, individually and collectively using multimodal processing techniques to build a rich content-based index for information retrieval, media segmentation, and media adaptation. Media processing is described next.

3.2 Media Processing

After the content is acquired, the media is processed to identify and segment relevant pieces of information. The details of the media segmentation techniques used are beyond the scope of this document and can be found in [1].

The results of the processing include high-level content features such as the locations of topic boundaries, topic keywords, and representative images for each topical content segment. Additionally, mid-level features are extracted as part of the processing and these are also maintained in the multimedia database. These include locations of scene boundaries, representative images for each scene (a.k.a. key frames), and an approximation of the dialog either in the form of closed caption text or the results of speech recognition (e.g., a word lattice or 1-best transcription.)

3.3 Content Repurposing

The high and mid-level content features described above can be exploited to enable the alerting system to support a wide range of device types [1]. Examples of media adaptation will be discussed in detail in the sections that follow and can be seen in Figures 7 and 8.

The interest profile obtained through the profile interface (described in more detail in Section 4) is used to find content that matches the user keywords. The device profile dictates the kind of content that is compatible with the user devices. The content is then repurposed depending on the destination device and is sent using the Alert Dissemination Engine (ADE), which is described next.

3.4 Alert Dissemination

The Alert Dissemination Engine (ADE) is a middleware solution that allows limited mobile devices to communicate with each other and to securely access corporate and Internet content/services. It consists mainly of *gateways* and *servers* and is an instance of the AT&T Enterprise Messaging Network (formerly known as iMobile-EE [2]).

Gateways handle protocol specific interfaces to mobile devices and perform authentication, device profiling and session management functions. Servers, that perform the task of verifying device accounts and scheduling are replicas and can be load balanced for enhanced reliability. The system operates as a dynamic environment, with gateways and servers discovering and adjusting their capabilities dynamically. Both gateways and servers can be dynamically added/removed to the system. Interconnecting the gateways and servers is a message based

communication infrastructure using both point-to-point and multicast models.

3.4.1 *Gateways and Devlets*

The platform provides gateways that host devlets (protocol interfaces) for a multitude of protocols: email, http, pager, voice, fax, SMS, instant messaging. Multimedia messaging is supported through the use of an MMS gateway that retrieves the picture/video content from the Media Storage and sends it to an MMS service provider through an HTTP connection (see Section 4.4 for details). Being the access points for both end user/devices and external systems, the gateways perform session initiation and management functions; within each user session, we maintain an associated delivery context.

The message oriented devlets are based on a messaging framework that covers the protocol specific implementations. It provides a clean separation between the application messaging protocols (for example 'pager') and the network access protocols used to deliver the messages (for 'pager' we usually have the choice of SMTP or SNPP). The framework offers support for message delivery tracking, selective retry policies, delivery channel monitoring, outbound to inbound message matching, and resource/bandwidth allocation. More details of the messaging framework can be found in [3].

In the context of the notification engine implemented within the platform, only some protocols are used as delivery channels, in particular, the message based asynchronous protocols: mail, SMS, instant messaging (Jabber, AIM, etc), pager, voice, and fax. Their main characteristics are that a recipient can be uniquely identified through a permanent protocol specific address: email address, phone number, etc. As a consequence, it is possible to perform a 'push' of a message towards the end recipient.

3.4.2 *Servers and Infolets*

The components that make up a Server's behavior are called *infolets*. Infolets implement the associated application logic and usually provide the access to one or more sources of information. Since the infolet output needs to be provided with respect to the delivery context established for the user session, the ADE offers a framework for information transcoding that can be used by the infolet provider but the ADE does not perform automatic transcoding itself. A particular class of infolets, called services, is dedicated for programmatically exposing functionality to external systems in contrast with other classes which provide content to the end user devices. The different components of the ADE are implemented as a set of Web Services operating on top of the infrastructure. For further details on this platform please refer to [3].

Device	Description	Messaging Capabilities
1) PPC 2002 Smartphone	Siemens SX56	GSM/GPRS/SMS
2) MMS Phone	SonyEricsson T610	GSM/GPRS/SMS/MMS
3) Alphanumeric Pager	Skytel pager	Email/Paging
4) Numeric Pager	Metrocall pager	Email/Paging
5) Blackberry	Blackberry 6710	GSM/GPRS/Email/SMS
6) Cell Phone	Nokia 3310	GSM/SMS
7) PPC 2003 Smartphone w/ WiFi	O2 Xda II	GSM/GPRS/SMS/WiFi/Bluetooth

Table 1 – User Device Descriptions

Figure 2 – User Devices

4 IMPLEMENTATION

MediaAlert supports the delivery of alerts with a range of media content including text, images, audio and video. The devices that are supported range from devices with limited display and processing capabilities such as pagers which can only handle limited text information or regular voice phones which can only receive phone calls, to PDA devices with video streaming capability. An assortment of devices currently supported by MediaAlert is shown in Figure 2. Device descriptions and messaging capabilities of these devices are shown in Table 1. Our purpose is to provide users with the flexibility to use any device. In the following sections, we present the implementation details of the prototype system.

4.1 USER PROVISIONING

The user can interact with the system in two ways. First, the user utilizes a Web interface to provision their devices and their interest profiles. Second, as new content is acquired that matches the user profile, the user will receive alerts on their selected devices. We will describe the provisioning component in this section. The alerting component will be discussed in the next section.

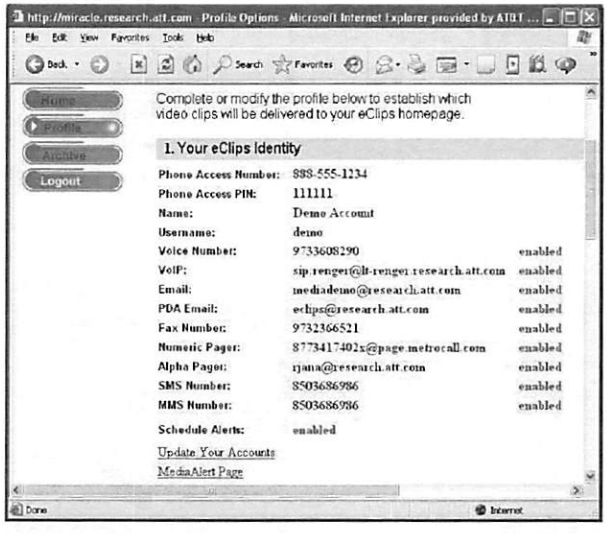


Figure 3 – User Device Profile

Devices/ Protocols	*Voice*	*VoIP*	*Desktop Email*	*PDA Email*	*Numeric Paging*	*Alpha Paging*	*SMS*	*MMS*
1) PPC 2002 Smartphone	X	X		X			X	
2) MMS Phone	X			X			X	X
3) Alphanumeric Pager						X		
4) Numeric Pager					X			
5) Blackberry	X		X	X		X	X	
6) Cell Phone	X						X	
7) PPC 2003 Smartphone w/ WiFi	X	X		X			X	X

Table 2 – Device Protocols

Figure 3 shows the user device profile Web page where the user provisions his or her devices. We maintain a distinction between the user contact list and the user notification list. Users can choose a subset of their devices from the contact list to be used for notification purposes (these are shown as "enabled" in Figure 3).

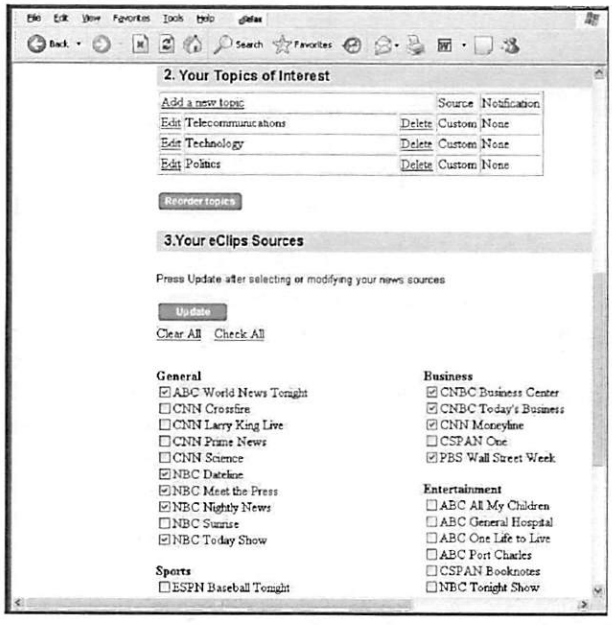


Figure 4 – User Interface Profile

As shown in Figure 3, the user can access the alert content via phone, VoIP, or other standard protocols. Audio can be delivered by making an alert call to a phone or to a VoIP (Voice over IP) client using SIP (session initiation protocol). Alternatively, the user can call a toll free Phone Access Number to hear the audio content to directly access the VXML interaction. In each of these cases, the Phone Access PIN is used to authenticate the user. Alerts can also be delivered to Email, Fax, Numeric or Alphanumeric Pager, and to SMS or MMS enabled devices. The protocols supported by the assortment of devices in Figure 2 are shown in Table 2. MediaAlert requires all user and device information to be pre-provisioned. Consequently, relevant user profile information is already available at the time of the alert generation in order to efficiently perform the dissemination.

Figure 4 shows the user interest profile Web page where the users provision their topics and associated keywords as well as the program sources for each topic. Each topic in the profile can have different keywords and can use a different subset of the available program sources. An alert is only sent if the keywords for a topic in the interest profile match content in the program sources associated with this topic. Keywords for topics are correlated against closed caption text, speech recognized audio segments and other metadata like EPG. This is described in more detail in [4].

4.2 ALERT GENERATION

After each TV news program is acquired and processed, the audio and video are transferred to the

Media Storage/Media Server which would ultimately stream the content to the devices that support streaming. The metadata and closed caption text are sent to the Index Server where the content is indexed.

Figure 5 – XML Content

Various approaches can be used to match new content with user profiles. In the current implementation, a task runs at specific times in the day for each user and identifies new content that matches the program sources and keyword requirements for each topic. The first step in this task is to flag the alert content for each user. All new content since the last alert time is written out to an XML file for each user as shown in Figure 5. This file includes data from the index and the data from post-processing. The above mentioned task not only extracts the relevant clips but also repurposes the content and interfaces with the ADE via Web Services to send out the alerts automatically. This is described in the next two sections.

Other approaches could entail real-time word spotting of the closed caption text as the content is being acquired. The alerts could effectively be sent immediately. Of course, the clipping segmentation and indexing algorithms would be less effective since it would not have the advantage of analyzing the entire broadcast.

4.3 Content Repurposing

To support different protocols and devices, we repurpose the content to match the device requirements in the profile. The content repurposing is accomplished by using the relevant information in the XML file in Figure 5. For instance, the alert fax contains the full text of the clip (textfull attribute) whereas all the other alerts use the synopsis text (text attribute).

Content/ Protocols	*Voice/ VoIP*	*Desktop Email*	*PDA Email*	*Fax*	*Numeric Paging*	*Alpha Paging*	*SMS*	*MMS*
1) Callback Number					x	x	x	x
2) Hyperlink to Video		x	x					x
3) Program Icon		x						x
4) Program Name	x			x				
5) Date	x	x	x	x		x		x
6) Topic	x	x	x	x		x		x
7) Duration	x	x	x	x				
8) Thumbnail		x		x				x
9) Synopsis Text		x	x			x	x	x
10) Full Text				x				
11) Audio	x							

Table 3 – Content vs. Device Protocols

Table 3 shows the various content elements used for each device protocol. Note that most of the content elements come directly from the XML file while others are derived from the information in the XML file. For instance, the synopsis text is stored in the "text" attribute of the "clip" element in the XML file as can be seen in Table 4.

Content	**XML Element**	**XML Attribute**
1) Callback Number	usercontent	emnnumber
2) Hyperlink to Video	clip	video
3) Program Icon	clip	banner
4) Program Name	clip	title
5) Date	clip	date
6) Topic	topic	name
7) Duration	clip	duration
8) Thumbnail	clip	thumbnail
9) Synopsis Text	clip	text
10) Full Text	clip	textfull
11) Audio	clip	[derived]

Table 4 – XML Content vs. Device Content

For text-only devices such as pagers and SMS devices, we provide the text content to the devices including a callback number. The text may be truncated to satisfy device requirements. Figure 6 shows the alert for an alphanumeric pager. The different elements from Table 3 are labeled accordingly. Note that for some devices/protocols, it is possible to send a hyperlink to the video. The media streaming server must be engineered to handle such video-on-demand requests based on the number of expected concurrent users. Our prototype uses Microsoft media streaming server. Other

video types can be created during the media adaptation process.

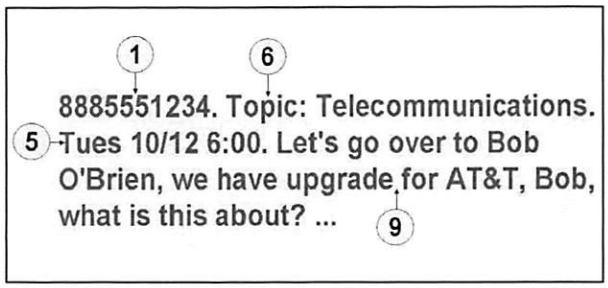


Figure 6 – Alphanumeric Pager Content

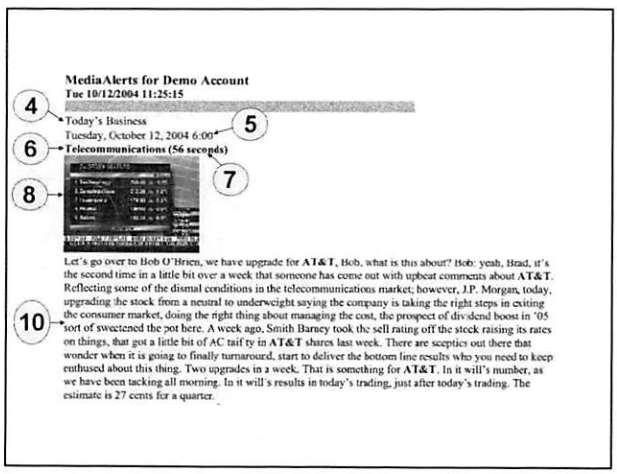


Figure 7 – Fax Content

The voice content is delivered to a phone or VoIP client via a VXML/SIP gateway via a remote dial option of the gateway. The audio alert content is created from the original video during media processing. The user can navigate the call using Touch-Tone commands or speech input. The prompts are played via TTS (text-to-speech) but the audio alert content is played back from the audio file. As previously discussed, the users can always dial the callback number at their preferred time instead of having the system call them.

For devices that can handle text and images, such as fax machines and MMS phones, a combined text and image representation is generated for delivery. Figure 7 shows a typical fax alert. For fax alerts, we generate html files that are sent to a fax broker which passes on the alert. For MMS phones, we compose the image portion and the text portion into an MMS message suitable for delivery by an MMS broker. Several clips can be concatenated and sent in one MMS message. For devices that can receive html formatted email, we send html email to the users directly. Figure 8 shows a typical desktop email alert. The thumbnail is a link to the video so clicking the thumbnail will stream the higher bit rate video.

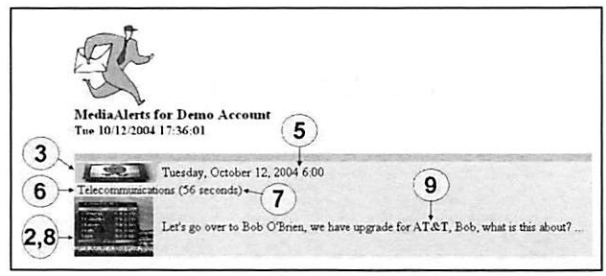


Figure 8 – Desktop Email Content

For video enabled mobile devices, such as PocketPC PDAs, we send a video link through the PDA email. The user can then stream the video by clicking on the link.

4.4 ALERT DELIVERY

The Desktop Email, PDA Email, Fax, Numeric Pager, Alphanumeric Pager, SMS, and MMS content is sent via the relevant ADE gateways. The Voice and VoIP content is delivered to the Phone Number or SIP address using the VXML/SIP gateway.

Since we use different devices which may be on different networks, the data transmission varies across different networks and protocols. We use two examples (MMS and Fax) to demonstrate the process involved in delivering alerts on various networks.

Each MMS phone registered for alert delivery is identified uniquely in our system by its phone number and is mapped to a particular user. Our MMS gateway interacts with an MMS service provider using an HTTP connection to their server. Typically, an MMS provider hosts an MMSC gateway and maintains connections to cellular carriers globally [5].

Unlike the other gateways in the ADE, the MMS gateway repurposes the content and sends out the alert. For the other gateways, the content is repurposed and is passed as plain or html text or as an html file to the destination gateways. In the case of the MMS gateway, the URL of the XML file is passed. The MMS gateway parses this XML file to locate all the images, text, and other content elements relevant to an MMS alert. It then retrieves these elements from the Media Storage and then sends the MMS message out to the MMS service provider.

For the fax gateway, we send email to a fax broker called eFax. First, we enhance the gateway to support email with attachments. Second, we solve the issue of accessing images inside the firewall. The fax gateway first receives html content, which contains image URLs inside our Intranet that eFax cannot directly access.

There are several ways to solve this, such as opening a port, using a reverse proxy, or converting html to other document formats. We decided to convert the html file to a PDF document that we can email to eFax as an attachment. The end result is a fax alert that contains text and images as shown in Figure 7.

5 System Evaluation

With our implementation, we evaluated the performance of the system and the user's experience. The performance was evaluated by measuring the execution time of various components under different conditions. Together with the user experience, the study helps us to better understand the system behavior and to improve and optimize the system.

5.1 Performance

Since MediaAlert consists of media processing and alert dissemination as two relatively independent components, we treat the two separately in the performance studies so that we can get more detailed data for each component. First, we discuss the performance on media processing. Then, we discuss the alert dissemination and message delivery component.

Content Processing Steps	Time (sec)	Percentage
Video transcoding	38	42%
Audio transcoding	6	6%
Caption text/JPEG processing	7	8%
Caption alignment with speech	39	43%
Other	1	1%
Total	**91**	**100%**

Table 5 – Content Processing Times (100 second video)

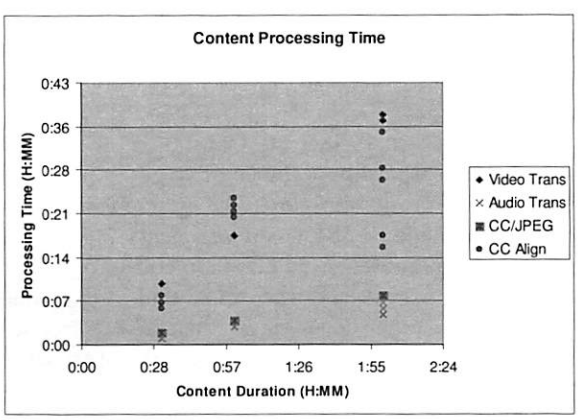


Figure 9 – Content Processing Time vs. Content Duration

5.1.1 *Media Processing*

Media processing occurs after the video acquisition and has two components: content processing and clip generation. The content processing applies to the full video and the clip generation applies to the portions of the video content that are sent as an alert.

The full video processing steps include: video transcoding, audio transcoding, closed caption (CC) and JPEG processing, and CC alignment with speech. Table 5 lists the elapsed time and the percentage of the total time in each step for a 100 second video. The total processing time is 91 seconds or 91% of the source video time which means that the current system can handle video streams in real-time. Figure 9 is the data from the acquisition of 17 video streams, from 30 minutes to 120 minutes. In this Figure, the transcoding time depends almost linearly on the length of the source video. Video transcoding takes about 31% of the video time; audio transcoding is 6%. CC/JPEG processing is 7%. The time for CC alignment processing depends on the content, ranging from 13% to 39% of the video time. The acquisition is done on a 1GHz dual-processor Pentium III for one broadcast input.

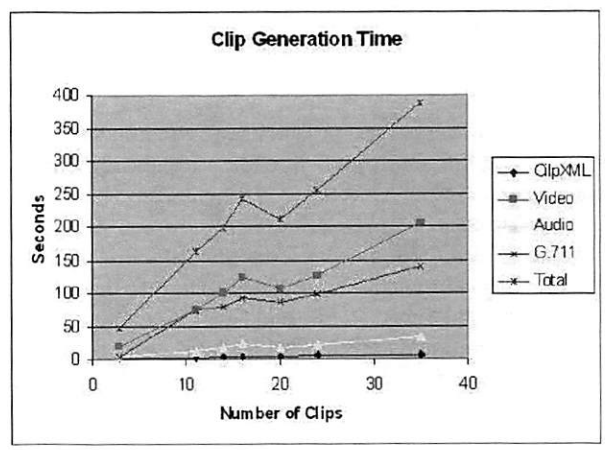


Figure 10 – Clip Generation Time vs. Number of Clips

Figure 10 is the timing information for clip generation which includes generating the clipping information XML file, extracting video and audio clips, and obtaining uncompressed audio (G.711) for use with the VXML audio playback. We have performed these tests by varying the number of clips from 3 to 35 by adjusting the time window of the search. Among the processing steps, clipping video is the most time consuming part. Uncompressing audio also takes a relatively long time because we use off-the-shelf tools directly without optimizing for our particular purpose. A portion of the same data is also available in Table 6, which gives the average clip duration and the average clipping time to process one clip. From this Table, we

see that clips with an average duration of 87 seconds long take 13 seconds to clip on average which is 15% of the clip duration time. This implies that the system is very efficient in processing clips. Compared with the content processing time, the clipping processing time is much less. The data was collected on a 2.4 GHz dual-processor Pentium machine.

Clips	Average clip duration (sec)	Average clip processing (sec)	Processing/ Duration
3	105.67	16.00	15%
11	96.00	14.82	15%
14	87.57	14.21	16%
16	87.63	15.19	17%
20	80.40	10.60	13%
24	79.63	10.63	13%
35	69.26	11.03	16%
Avg.	86.59	13.21	15%

Table 6 – Clip Duration/Processing Times

5.1.2 Alert Dissemination

In this section, we evaluate the performance of the alert dissemination component system for email alerts. Figures 11 and 12 show the measurements at the client side from the time when the alert injection requests are sent to the time when the EMN server responses are received. This is the time that is required to satisfy an alert specification and output a time slotted schedule. This schedule determines when each alert is disseminated. The client machine simulates alert requests with multiple threads at the recipient mailboxes.

We used an EMN testing framework, which is capable of simulating multiple threads of alert requests. In Figures 11 and 12, the number of threads is configured as 1, 2, 4, 8 and 16. The number of endpoints (number of recipients of an alert) that each client thread generates varies from 32, 64, 128, 256, 512 to 1024. To show performance measurement results of these configurations, we use representative endpoints of 32, 64 and 128 as shown in Figure 11. The results with endpoints from 256 to 1024 are shown in Figure 12. For each figure, the five white bars on the left indicate the number of threads being used. From left to right, the number of threads is 1, 2, 4, 8 and 16. These white bars are for the cases when there is only one EMN server. Similarly the five dark bars indicate the cases when there are two EMN servers. Note that due to the distributed nature of the platform it is possible to load balance the alert processing via identical EMN servers and therefore accommodate a larger number of users. As a rule of thumb, new servers should be started automatically if the sustained load exceeds a certain safety threshold, thereby maintaining scalability. The increasing number of endpoints shows the workload change. We put the one-server and two-server cases side by side for ease of comparison. In this measurement, the client uses a 1.4GHz dual-processor Pentium machine running a Linux 2.4 kernel. Gateways and EMN Servers use 2.4GHz quad-processor Pentium machines with a Linux 2.4 kernel.

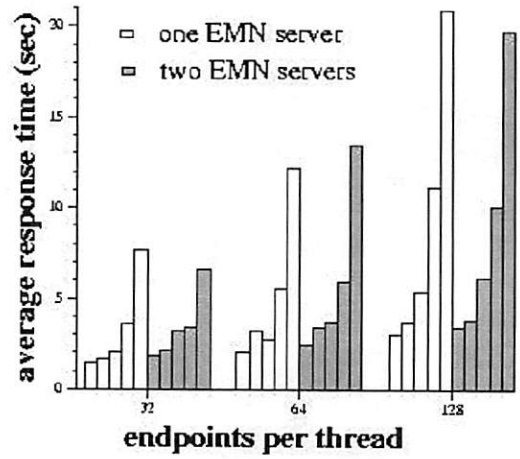


Figure 11 – Average Server Response Time through EMN ADE: Part I

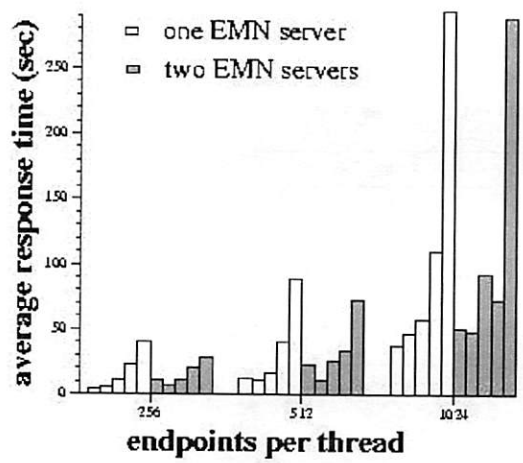


Figure 12 – Average Server Response Time through EMN ADE: Part II

Figures 11 and 12 indicate that the server response time increases when the number of requests increases. As we can see from the figures, the system can process alerts that include 16384 email recipients (16 threads with 1024 endpoints each) in less than 300 seconds with either one or two servers. The actual end-to-end email dissemination time took less than 700 seconds. However, even for large requests, the two-server configuration with two JMS queues has similar performance as the one server case. This indicates that the system bottleneck is not at the server engines.

Because all the requests need to access the common Oracle database, the database access appears to be the bottleneck in the system.

Based on the data we collected, media processing tends to take longer than alert dissemination, especially for a small number of users. To balance the system performance, we need to first reduce the media processing time. The number of Broadcast News channels is limited by the number of available channels but the number of users can be increased arbitrarily. To be cost effective, a good balanced system relies on the scale of the service.

The final delivery time from the server to an end device, which depends on the access mechanism available on the device, can also add a significant delay to the alerting process. Let's take the delivery of MMS messages to a mobile phone as an example. In one experiment, a one-character MMS message took 49 seconds. For messages with multiple text and picture components, it took 90 seconds for 20KB of data and 114 seconds for 40KB of data. The time fluctuations between different runs can be substantial. Sometimes, MMS message delivery can take more than 10 minutes. Because the messages have to traverse different networks which are beyond our control (the MMSC handles messages in a store and forward mechanism), the delay can be very long and unpredictable. In such extreme cases, we need a mechanism to detect the abnormality immediately and to switch to an alternative device for the user.

5.2 User Experience with MediaAlert

In this section, we discuss our preliminary experience with MediaAlert on several mobile devices.

As shown in Table 2, the Blackberry 6710 is a versatile GSM/GPRS mobile device with voice, email, SMS, and paging capabilities. New email is pushed automatically to a Blackberry device without the mobile user having to access a mailbox explicitly. Also, the Blackberry can handle large amounts of email text by simply requesting for more email from the Microsoft Exchange server. This makes the Blackberry an ideal device for receiving comprehensive text alerts.

Initially, we included the callback number only for protocols with limited text abilities (such as paging and SMS). For example, we cannot send more than 140 characters (and sometimes 100) in an SMS message to most phones. The callback number is included so that users can call back to get the complete audio clip. On the Blackberry 6710, any sequence of digits (such as 9735551212) that looks like a phone number is clickable and the clicking initiates a call to that number. This is true for both SMS messages and email messages received on that device. This feature makes it very convenient for retrieving audio alerts and has increased our voice usage of the Blackberry 6710. It has also prompted us to add a callback number to email alerts. Similarly for an MMS message, the MMSC sends the user a notification that a new message is waiting. The receiver can then download the message immediately or download it later (user pull rather than a user push request). Although an MMS message can encompass a wide range of content types, it is a logical extension of SMS, making it easily adoptable for today's generation of mobile users. Another advantage of MMS for this kind of alert is that the message is delivered as a single multimedia message and not as a text message with attachments. This minimizes the steps that the user has to take to retrieve the content.

On the recent Xda II device (Smartphone with PocketPC 2003, WiFi, and Bluetooth), we experimented with low bit rate streaming video alerts using the Microsoft Media Player. Due to copyright issues, we conducted our experiments on an internal Lab server. A lower bit rate of 150 kbps was used for streaming content to these PDAs. Overall, the mobile user is able to watch the streamed video comfortably without much data loss. As most 2.5G and 3G wireless networks are still limited in bandwidth, we expect the retrieval of high quality video alerts to become feasible first on WiFi networks. Hopefully, the convergence of 3G and WiFi/WiMax on a new generation of cell phones will allow the users to retrieve videos with varying degrees of quality depending on the cost/network availability.

The above comments on user experience are based on *scheduled alerts*. Other types of alerts can be handled in a similar fashion or treated differently. Further user experience study would be needed to ensure that we would be meeting user needs. It is also possible in the future to extend the user profile to incorporate other attributes like location, presence and context.

6 Related Work

Palmer et al describe a system [7] where speech recognition and machine translation techniques are applied to TV news programs. This work goes into multi-lingual capabilities where real-time information is detected and matched against subscribed keywords from English and Arabic news sources. While the demonstration explains how to extract interested content automatically and display it in a cross-lingual environment, content repurposing and the problems of alerting mobile users are not addressed. Sumiya et al. present a system [8] to bring broadcast news content to the Web by using metadata with a zooming feature to alter the level of detail being viewed. This work focuses on repurposing the content for Web browsing. There has also been some research conducted in topic detection

from TV programs, in particular, robustness in topic boundary identification amidst transcription errors [9].

Google Alerts are email updates of the latest relevant Google results (Web, news, etc.) based on the user's choice of query or topic. Our proposed platform takes this one step further to incorporate live TV broadcast feeds. It will be an interesting experiment to see the effectiveness of MediaAlert when compared to GoogleAlerts [10] or CNN Alerts [11] when monitoring a developing news story. The key features that differentiate our system from other commercial services are automated video content acquisition and monitoring, topic segmentation, and media content adaptation for mobile devices. In addition, we also support speech recognition (using a two hundred thousand word vocabulary automatic speech recognizer). There are a number of products that offer mobile-enterprise focused alerting and notification. IBM has also announced an additional package to its Websphere Everyplace Access Suite [12]. These additions depend on vendor specific products like Sametime (now Lotus Instant Messaging) and presence platforms [13].

Microsoft has recently announced .NET alerts - a message and notification service that can be used to deliver customer communication to desktops, cellular phones and personal digital assistants [14]. This solution is more powerful in the Intranet since it makes use of Microsoft related products deployed within an enterprise, for example, Microsoft Active Directory, SQL Server notification services and .NET Framework. MediaAlert presents an extensible architecture ready for integration with existing enterprise software in a standardized and vendor agnostic manner. In our work, we have concentrated on the notification aspects, trying to offer an open generic common interface to alert management software that incorporates the business logic, workflow and decision aspects.

The media processing work in our approach is based on earlier work [15]. Our previous work [16] aimed to have the ability to transfer emergency video clips captured from a mobile user to other relevant users in a timely fashion. In this case, there was no media analysis, voice or text processing involved.

7 Discussion

A whole new category of converged mobile devices will soon be upon us. PDAs will be combined with cell phones and radios and televisions. We will not predict the winners and losers of this device war but we believe that there will be more need for all types of alerts in order to best utilize the capabilities of these converged devices. Unless the system or network takes over more of the work, the user will be overwhelmed with choices and inhibited by the User Interface to such an extent that the entire concept of converged devices will falter. These combined devices will also enable users and designers to skirt the edges of copyright law in ways that will allow new services to exist. In the realm of mobile devices and expanding on the concepts of immediate and future alerts, we can discuss the convergence of cell phones and digital television as it impacts alerting services.

Next generation cell phones will enable users to receive the content from HDTV broadcasts [6]. Wireless handsets could receive hundreds of high-definition digital channels, not over the cell network but from the HDTV broadcast network or from a separate satellite network that could carry hundreds of digital channels. The article in [6] claims and we concur that this technology would be "event-driven". We expect this capability to open up new "event-driven" alerting possibilities for the mobile user, beyond the simple notion in the article about watching specific sports or news shows.

We anticipate a converged future for devices, where they will not necessarily be all-in-one devices but they will all communicate and interoperate. As an example scenario, imagine that an alert is sent for a particular event on news or TV. If the event occurs, the user would be notified and could instantly watch the clip of interest. The main advantage of this scenario over the previous alerting scenario is in the realm of copyrights. If we wanted to deliver a service that would create video clips of interest for a user, we would be required to get licenses from all the providers of multimedia material. That proves to be a daunting task and has held up much of the useful online multimedia applications. If instead of having to capture and rebroadcast the material, we could simply monitor it and as we locate it send that information out to users who could then watch it live. In this way, we can provide the alerting service without violating copyright laws. This scenario opens up other possibilities for the user. As an example of the converged future, there could be a video recording device at home that could be notified over the Internet that this segment was of particular interest and needs to be recorded. The difference between this and an automatic program recorder is that our service would only be recording the specific clips that were of interest to the user. Once they were recorded at the user's home, that user could watch them at leisure. The user could even have the clips streamed from the video recording device directly to the video-enabled mobile handset. We envision getting support from the handset makers and the cell service providers because they are both desperate to generate new service revenue and have invested billions of dollars in upgrading to high speed digital networks. This would certainly have appeal to the handset makers as a way to enhance the capabilities of their phones. The streaming to the device

would be of great interest to the cell service providers as a way to generate new revenue, particularly if this were set up as an emergency alerting system for public service workers like police, fire and first aid.

8 CONCLUSIONS

An automated system for multimedia content monitoring and alerting was described. Unlike existing systems that rely on manually generated clips/stories, this system uses multimodal story segmentation algorithms to find and isolate short relevant segments of video within a video program. Moreover, it relies on multimedia processing techniques to repurpose the content for delivery to a range of mobile devices with a wide range of presentation capabilities from text-only to full-motion video. The repurposing process is not only aimed at producing a representation of the information that accommodates the limitations of the device at hand, but is also aimed at creating alternative presentations that significantly reduce the amount of bandwidth needed to deliver the information.

Targeting this application mainly for mobile devices and using it to generate alerts require higher selectivity in choosing the information and better isolation of the information. Such high selectivity is critical for preventing the generation of false or trivial alerts. This can be achieved by a combination of better information processing/retrieval techniques and good judgment on the part of the user in providing the right combination of keywords and phrases for each topic of interest. Our experience indicates that imposing additional proximity constraints in the retrieval process is an effective way for increasing the relevance of extracted content and reducing the possibility of false alerts. Another issue is the generation of multiple alerts with very similar information from different sources. We are working on utilizing effective similarity measures to detect and eliminate such cases.

The combination of automatic media monitoring and alerting not only provides an effective system for timely delivery of personalized information and timely business information, but is also an effective way for automatically discovering and delivering security related information.

MediaAlert has been designed to be a carrier grade solution both in terms of architecture scalability and flexibility to innovate and deploy new services rapidly. The media processing engine can ingest a large number of simultaneous real time broadcast quality feeds while the dissemination engine can handle a large number of concurrent alerts to meet stringent timing requirements.

REFERENCES

[1] D. Gibbon, L. Begeja, Z. Liu, B. Renger, and B. Shahraray, "Multimedia Processing for Enhanced Information Delivery on Mobile Devices," *Emerging Applications for Wireless and Mobile Access,* MobEA II, New York, May 18, 2004.

[2] Y. Chen, H. Huang, R. Jana, T. Jim, S. Jora, R. Muthumanickam, and B. Wei, "iMobile EE – An Enterprise Mobile Service Platform," *Wireless Networks,* Vol. 9, No. 4, pp. 283-297, July 2003.

[3] S. Jora, R. Jana, Y. Chen, M. Hiltunen, T. Jim, H. Huang, A. Singh, R. Muthu, "An alerting and notification service on the AT&T Enterprise Messaging Network," *Proceedings of IASTED – Internet and Multimedia*, Feb 21-23, Grindelwald, Switzerland, 2005.

[4] L. Begeja, D. Gibbon, K. Huber, A. Lee, Z. Liu, B. Renger, B. Shahraray, M. Zalot, G. Zamchick, "eClips: Customized Video Clips," talk at WebSummit 2001.

[5] 3GPP2 Multimedia Messaging System: MMS Specification Overview http://www.3gpp2.org/Public_html/specs/X.S0016-000-A.pdf.

[6] NY Times, "Coming Soon to Your Pocket: High-Definition TV Phones," October 21, 2004.

[7] D. Palmer, P. Bray, M. Reichman, K. Rhodes, N. White, A. Merlino, and F. Kubala. "Multilingual Video and Audio News Alerting," *Human Language Technology Conference,* May 2004.

[8] K. Sumiya, M. Munisamy, and K. Tanaka, "TV2Web: Generating and Browsing Web with Multiple LOD from Video streams and their Metadata," *The Thirteenth International World Wide Web Conference*, May 2004.

[9] Richard Schwartz, Toru Imai, Francis Kubala, Long Nguyen, and John Makhoul, "A maximum likelihood model for topic classification of broadcast news," in *Proceedings of Eurospeech '97*, Rhodes, Greece, Sept. 1997, pp. 1455–1458.

[10] Google Alerts - http://www.google.com/alerts

[11] CNN Alerts - http://www.cnn.com/EMAIL/

[12] T. Olavsrud, "Websphere adds wireless notifications, messaging", InstantMessagingPlanet.com, May 2003, http://www.instantmessagingplanet.com/wireless/article.php/2201571

[13] Sametime – Lotus Instant Messaging, http://www.lotus.com/products/lotussametime.nsf/wdocs/homepage

[14] Microsoft .NET alerts v6.0 - http://msdn.microsoft.com/msdnmag/issues/03/12/XMLFiles/default.aspx

[15] L. Begeja, B. Renger, D. Gibbon, K. Huber, Z. Liu, B. Shahraray, R. Markowitz, P. Stuntebeck,

"eClips: A New Personalized Multimedia Delivery Service," *Journal of the Institution of British Telecommunications Engineers (IBTE)*, April-June 2001.

[16] B. Wei, Y. Chen, H. Huang, and R. Jana, "A Multimedia Alerting and Notification Service for Mobile Users," *Emerging Applications for Wireless and Mobile Access,* MobEA II, New York, May 18, 2004.

[17] Y. Chen, H. Huang, R. Jana, S. John, S. Jora, A. Reibman, and B. Wei, "Personalized Multimedia Services Using a Mobile Service Platform," *IEEE Wireless Communication and Networking Conference,* Orlando, FL, March 2002.

[18] S. Jun, Y. Rong, S. Pei, and S. Song, "Interactive Multimedia Messaging Service Platform," *Proc. ACM Multimedia.* pp. 464-465, Berkeley, CA, November 2003.

[19] Eric Manning, "The Utility Model for Multimedia Servers and Networks: A Retrospective," *Keynote Address, ICCIT,* December 2002.

[20] Microsoft 9 Series Digital Rights Management, http://www.microsoft.com/windows/windowsmedia/9series/drm.aspx.

[21] Microsoft Windows Media-on-Demand Producer, http://www.microsoft.com/technet/prodtechnol/netshow/downloads/wmodp.mspx.

[22] D. Gibbon, L. Begeja, Z. Liu, B. Renger, and B. Shahraray, "Creating Personalized Video Presentations using Multimodal Processing," *Handbook of Multimedia Databases,* Edited by Borko Furht, CRC Press, pp. 1107-1131, June 2003, ISBN 0-8493-7006-X.

[23] Han Chen, Kai Li, and Bin Wei, "Memory Performance Optimizations For Real-Time Software HDTV Decoding," *Journal of VLSI Signal Processing, Special Issue on Media and Communication Applications on General Purpose Processors: Hardware and Software Issues,* accepted for publication, 2005.

[24] Gibbon, D. *et al* (1999). "Browsing and Retrieval of Full Broadcast-Quality Video," *Packet Video,* NY, NY, April 25.

Cracking the Bluetooth PIN*

Yaniv Shaked and Avishai Wool

School of Electrical Engineering Systems,
Tel Aviv University, Ramat Aviv 69978, ISRAEL
shakedy@eng.tau.ac.il, yash@acm.org

Abstract

This paper describes the implementation of an attack on the Bluetooth security mechanism. Specifically, we describe a passive attack, in which an attacker can find the PIN used during the pairing process. We then describe the cracking speed we can achieve through three optimizations methods. Our fastest optimization employs an algebraic representation of a central cryptographic primitive (SAFER+) used in Bluetooth. Our results show that a 4-digit PIN can be cracked in less than 0.3 sec on an old Pentium III 450MHz computer, and in 0.06 sec on a Pentium IV 3Ghz HT computer.

1 Introduction

1.1 Background

Bluetooth, a technology used for short range fast communications, has quickly spread worldwide. Bluetooth technology is used in a large set of wired and wireless devices: mobile phones, PDA's, desktop and mobile PC's, printers, digital cameras, and dozens of other devices. Being wireless, Bluetooth is potentially vulnerable to many attacks. It is very difficult to avoid Bluetooth signals from leaking outside the desired boundaries. The possible damage of a successful wireless attack starts with the ability to eavesdrop on the data transferred during the communication of two devices, and ends with the ability to fully impersonate other devices.

The Bluetooth technology has a significant security component, which includes key management, authentication and secrecy. However, the security of the whole system relies on the user's choice of a secret Personal Identification Number (PIN) - which is often much too short. Moreover, the Bluetooth designers invented several new cryptographic primitives, which were incorporated into the system. Cryptographers consider fielding new primitives to be risky, because new cryptography is less tested and may contain hidden flaws. Furthermore, Bluetooth is designed for short-range communication (nominal range of about 10m). This short-range is perceived as a security feature, since an attacker is supposed to be quite near the attack target - but recent history with IEEE 802.11 has shown that effective range-extenders can be built very cheaply [Reh03]. Finally, as Bluetooth gains popularity on PDAs and laptops, the information that lures attackers grows from cell-phone address books to valuable corporate data.

1.2 Related work

A number of crypt-analytical results regarding Bluetooth [HN99, FL01, Flu02, Kra02, Arm02, LV04, LW05] have appeared over the last five years. Most of the work has focused on the lowest level cipher, called E_0. This is a completely new cipher, designed specifically for Bluetooth. The current state of the art is that no practical attacks, with current technology, have surfaced, *yet*. However, it is already clear that the security of the cipher is much less than claimed: although E_0 uses 128-bit keys, its effective security is no more than an 84-bit system (approximately). If E_0 were to be used outside of the Bluetooth system, and allowed to produce a stream of several million bits, then [LV04] shows E_0 to be effectively a 39-bit system - which would make it much too weak for use. These are worrisome, if not yet fatal, developments.

As for a security analysis of the system as a whole, much less has been done. The most significant work so far is [JW01], which identified some weaknesses. Over the last two years some hacker tools are starting to emerge (with colorful names such as "bluesnarfing" [Lau03], "bluejacking" [Blu04], and "redfang" [Whi03]).

The work closest to ours was recently done by O. Whitehouse, independently, and announced at the

*Supported in part by a grant from Intel Corporation.

Term	Explanation
PIN	Personal Identification Number. The PIN code is 1-8 bytes long (8-128 bits). However, most devices use PIN sizes of 4 decimal digits.
BD_ADDR	Each Bluetooth device has a 48 bit unique address that is called the Bluetooth Device Address.
Pairing	The process in which two (or more) Bluetooth devices hook up to create a shared secret value called K_{init}. The K_{init} forms the basis for all future Bluetooth negotiations between these two devices.

Table 1: List of terms

CanSecWest '04 conference [Whi04]. His work contains a survey of many aspects of Bluetooth security. However, as far as PIN cracking goes, the author only describes the attack framework, with rough time estimates. Precise technical details of the attack (beyond the presentation slides) have not been published. Unlike our work, the author apparently did not implement a PIN-cracking program. Thus we believe that our implementation, measurements, and our optimization methods, are all novel.

1.3 Contribution

In this paper we introduce a passive attack, in which an attacker can find the PIN used during the Bluetooth pairing process. We then describe implementations of this attack, using three optimizations methods. For this purpose we wrote a special-purpose Bluetooth security suite from scratch. Our fastest optimization employs an algebraic representation of a central cryptographic primitive (SAFER+) used in Bluetooth. Our results show that a 4-digit PIN can be cracked in less than 0.3 sec on an old Pentium III 450MHz computer, and in 0.06 sec on a Pentium IV 3Ghz HT computer. We then sketch an additional attack that can force the Bluetooth devices to repeat the pairing process and make them vulnerable to the first attack.

Organization: In Section 2 we give an overview of Bluetooth security, focusing on the Bluetooth pairing and authentication mechanism. Section 3 describes the passive attack, in which an attacker can find the PIN used during the pairing process. Section 4 contains a description of five versions implementing such an attack, with their respective performance figures. Section 5 sketches the additional attack, which can force two devices to repeat the pairing process. Section 6 presents countermeasures that will reduce the probability of being subjected

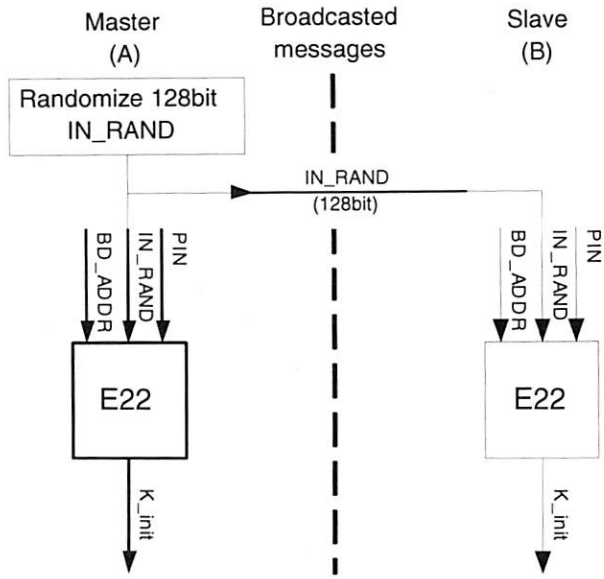


Figure 1: Generation of K_{init} using E_{22}

to both attacks and the vulnerability to these attacks, and we conclude our work in Section 7.

2 Overview of Bluetooth Security

A detailed specification of Bluetooth security mechanisms can be found in part H of Vol 2 of [Blu03]. A list of terms used repeatedly in this paper is given in Table 1.

This papers deals with the mechanisms used in Bluetooth Security Mode 3: The Link-level security mode. In this mode, a Bluetooth device will initiate security measures before a channel is established. This is a built-in mechanism, that is used regardless of the application layer security that may also be used. In security mode 3 terminology, establishing a channel between two Bluetooth devices is called pairing or bonding.

2.1 The Bluetooth pairing & authentication process

The Bluetooth initialization procedures consists of 3 or 4 steps:

1. Creation of an initialization key (K_{init}).
2. Creation of a link key (K_{ab}).
3. Authentication.

After the 3 pairing steps are completed, the devices can derive an encryption key to hide all future communication in an optional fourth step.

Before the pairing process can begin, the PIN code must be entered into both Bluetooth devices. Note that in

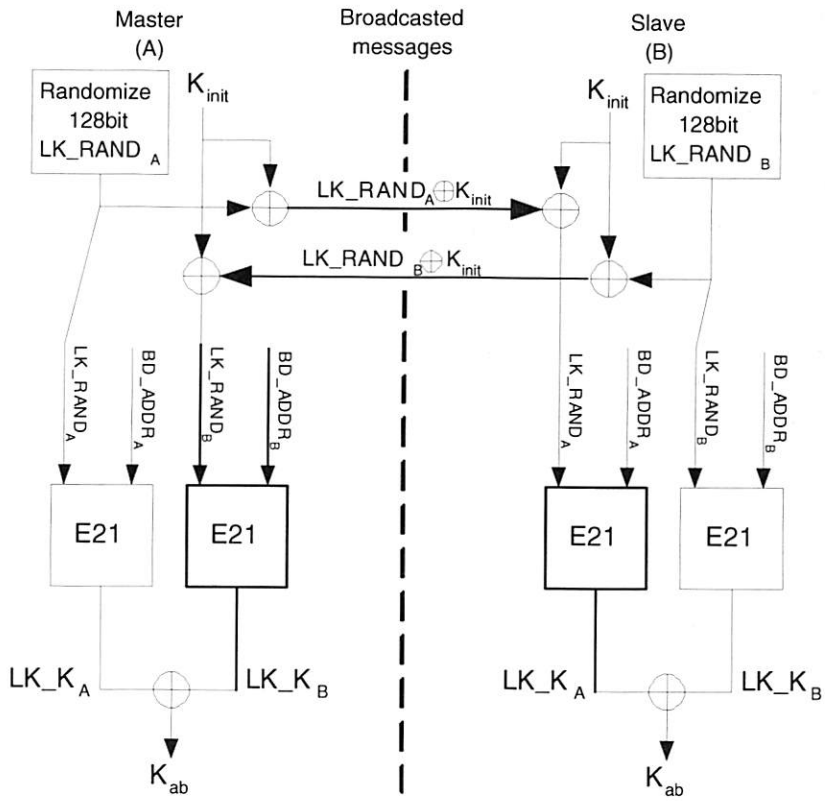


Figure 2: Generation of K_{ab} using E_{21}

some devices (like wireless earphones) the PIN is fixed and cannot be changed. In such cases, the fixed PIN is entered into the peer device. If two devices have a fixed PIN, they cannot be paired, and therefore cannot communicate. In the following sections we go into the details of the steps of the pairing process.

2.1.1 Creation of K_{init}

The K_{init} key is created using the E_{22} algorithm, whose inputs are:

1. a *BD_ADDR*.
2. the PIN code and its length.
3. a 128 bit random number *IN_RAND*.

This algorithm outputs a 128 bit word, which is referred to as the initialization key (K_{init}).

Figure 1 describes how K_{init} is generated using E_{22}. Note that the PIN code is available at both Bluetooth devices, and the 128 bit *IN_RAND* is transmitted in plaintext. As for the *BD_ADDR*: if one of the devices has a fixed PIN, they use the *BD_ADDR* of the peer device. If both have a variable PIN, they use the PIN of the slave device that receives the *IN_RAND*. In Figure 1, if both devices have a variable PIN, BD_ADDR_B shall be used. The Bluetooth device address can be obtained via an inquiry routine by a device. This is usually done before connection establishment begins. A detailed explanation of the inner design of E_{22} can be found in Appendix B.1.

This initialization key (K_{init}) is used only during the pairing process. Upon the creation of the link key (K_{ab}), the K_{init} key is discarded.

2.1.2 Creation of K_{ab}

After creating the initialization key, the devices create the *link key* K_{ab}. The devices use the initialization key to exchange two new 128 bit random words, known as LK_RAND_A and LK_RAND_B. Each device selects a random 128 bit word and sends it to the other device after bitwise xoring it with K_{init}. Since both devices know K_{init}, each device now holds both random numbers LK_RAND_A and LK_RAND_B. Using the E_{21} algorithm, both devices create the *link key* K_{ab}. The inputs of E_{21} algorithm are:

1. a *BD_ADDR*.
2. The 128 bit random number *LK_RAND*.

Note that E_{21} is used twice is each device, with two sets of inputs. Figure 2 describes how the link key K_{ab} is created. A detailed explanation of the inner design of E_{21} can be found in Appendix B.2.

2.1.3 Mutual authentication

Upon creation of the link key K_{ab}, mutual authentication is performed. This process is based on a challenge-response scheme. One of the devices, the verifier, randomizes and sends (in plaintext) a 128 bit word called AU_RAND_A. The other device, the claimant, calculates a 32 bit word called *SRES* using an algorithm E_1. The claimant sends the 32 bit *SRES* word as a reply to the verifier, who verifies (by performing the same calculations) the response word. If the response word is successful, the verifier and the claimant change roles and repeat the entire process. Figure 3 describes the process of mutual authentication. The inputs to E_1 are:

1. The random word AU_RAND_A.
2. The link key K_{ab}.
3. Its own Bluetooth device address (BD_ADDR_B).

A detailed explanation of the inner design of E_1 can be found in Appendix B.3.

Note that as a side effect of the authentication process, a 96 bit word called ACO is calculated by both peers. This word is optionally used during the creation of the encryption key. The creation of this encryption key exceeds our primary discussion and shall not be described in this paper.

2.2 Bluetooth cryptographic primitives

As we described above, the Bluetooth pairing and authentication process uses three algorithms: E_{22}, E_{21}, E_1. All of these algorithms are based on the SAFER+ cipher [MKK98], with some modifications. Here we describe features of SAFER+ that are relevant to our attack.

2.2.1 Description of SAFER+

SAFER+ is a block cipher [MKK98] with a block size of 128 bits and three different key lengths: 128, 192 and 256 bits. Bluetooth uses SAFER+ with 128 bit key length. In this mode, SAFER+ consists of:

1. KSA - A key scheduling algorithm that produces 17 different 128-bit subkeys.
2. 8 identical rounds.
3. An output transformation - which is implemented as a xor between the output of the last round and the last subkey.

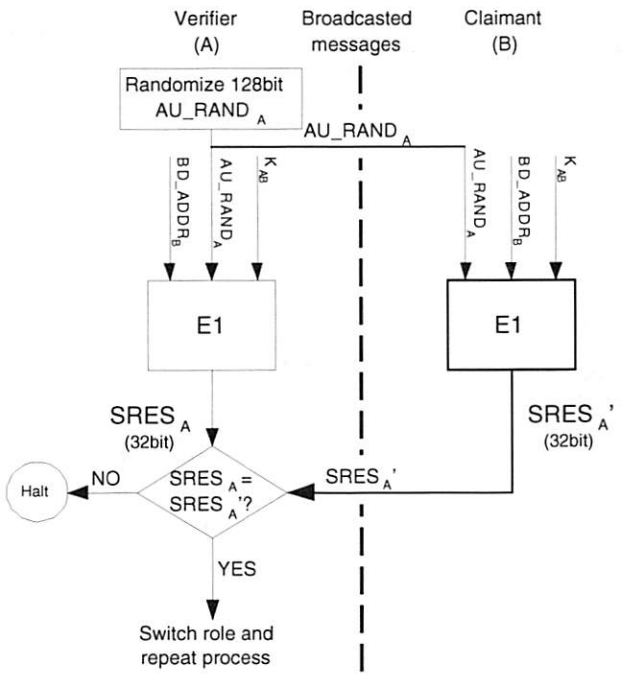


Figure 3: Mutual authentication process using E_1

Figure 4 describes the inner design of SAFER+, as it is used in Bluetooth.

The key scheduling algorithm (KSA)

The key scheduling algorithm used in SAFER+ produces 17 different 128-bit subkeys, denoted K_1 to K_{17}. Each SAFER+ round uses 2 subkeys, and the last key is used in the SAFER+ output transformation. The important details for our discussion are that in each step of the KSA, each byte is cyclic-rotated left by 3 bit positions, and 16 bytes (out of 17) are selected for the output subkey. In addition, a 128 bit bias vector, different in each step, is added to the selected output bytes. The entire process of key scheduling is detailed in Appendix A.1.

The SAFER+ Round

As depicted in Figure 4, SAFER+ consists of 8 identical rounds. Each round calculates a 128 bit word out of two subkeys and a 128 bit input word from the previous round. The central components of the SAFER+ round are the 2-2 Pseudo Hadamard Transform (PHT), the Armenian Shuffles, and the substitution boxes denoted "**e**" and "**l**".

The Pseudo Hadamard Transform takes two input bytes and produces two output bytes, as follows:

$$PHT[a, b] = [(2a + b) \bmod 256, (a + b) \bmod 256]$$

The Armenian Shuffle is a permutation of 16 bytes. See Figure 5 for the Armenian shuffle order.

The substitution boxes "**e**" and "**l**" are non-linear, both replace an input byte with an output byte. Their imple-

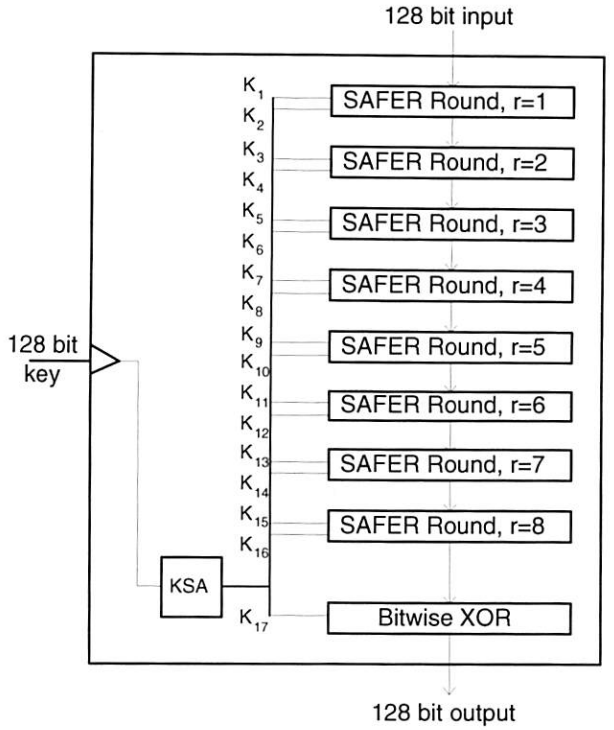


Figure 4: Inner design of SAFER+

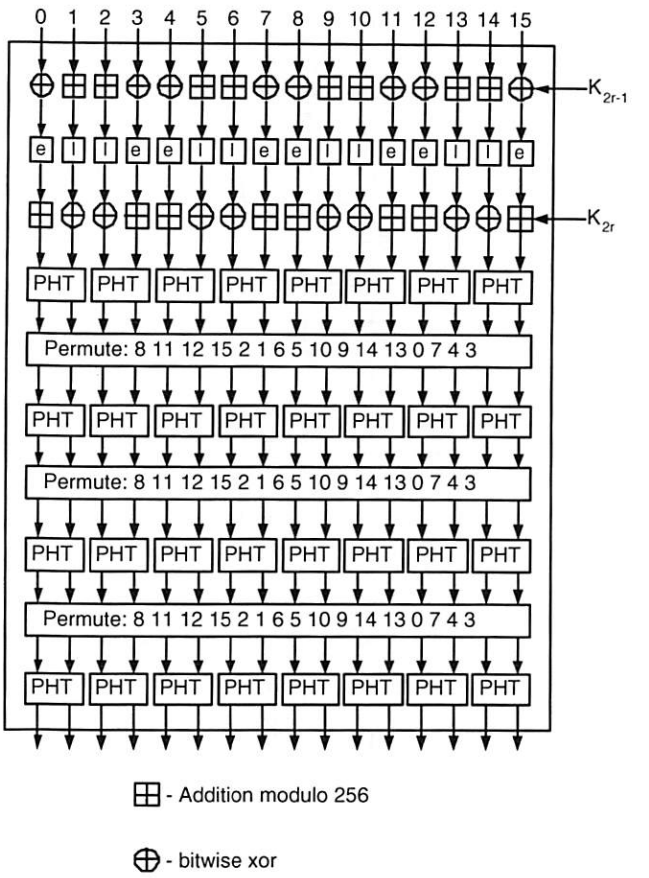


Figure 5: Structure of one SAFER+ round

mentation is given in equations (1) and (2):

$$e(x) = (45^x \pmod{257}) \bmod 256 \quad (1)$$

$$l(x) = y \ \ s.t. \ \ e(y) = x \quad (2)$$

Figure 5 describes the structure of one SAFER+ round.

3 Bluetooth PIN Cracking

3.1 The Basic Attack

Assume that the attacker eavesdropped on an entire pairing and authentication process, and saved all the messages (see Table 2). The attacker can now use a brute force algorithm to find the PIN used. The attacker enumerates all possible values of the PIN. Knowing *IN_RAND* and the *BD_ADDR*, the attacker runs E_{22} with those inputs and the guessed PIN, and finds a hypothesis for K_{init}. The attacker can now use this hypothesis of the initialization key, to decode messages 2 and 3. Messages 2 and 3 contain enough information to perform the calculation of the link key K_{ab}, giving the attacker a hypothesis of K_{ab}. The attacker now uses the data in the last 4 messages to test the hypothesis: Using K_{ab} and the transmitted AU_RAND_A (message 4), the attacker calculates *SRES* and compares it to the data of message 5. If necessary, the attacker can use the value of messages 6 and 7 to re-verify the hypothesis K_{ab} until the correct PIN is found. Figure 6 describes the entire process of PIN cracking.

#	Src	Dst	Data	Length	Notes
1	A	B	*IN_RAND*	128 bit	plaintext
2	A	B	LK_RAND_A	128 bit	XORed with K_{init}
3	B	A	LK_RAND_B	128 bit	XORed with K_{init}
4	A	B	AU_RAND_A	128 bit	plaintext
5	B	A	*SRES*	32 bit	plaintext
6	B	A	AU_RAND_B	128 bit	plaintext
7	A	B	*SRES*	32 bit	plaintext

Table 2: List of messages sent during the pairing and authentication process. "A" and "B" denote the two Bluetooth devices.

Note that the attack, as described, is only fully successful against PIN values of under 64 bits. If the PIN is

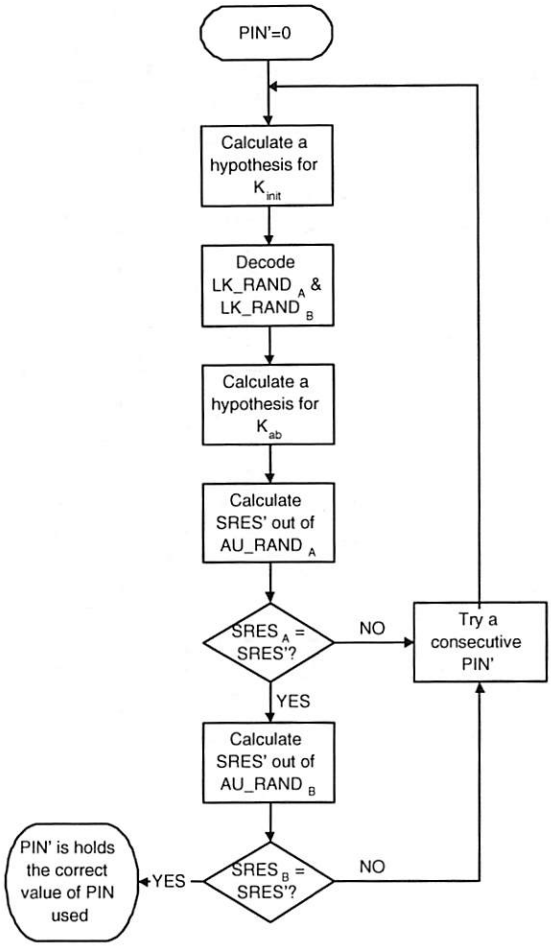


Figure 6: The Basic Attack Structure.

longer, then with high probability there will be multiple PIN candidates, since the two *SRES* values only provide 64 bits of data to test against. A 64 bit PIN is equivalent to a 19 decimal digits PIN.

4 Implementation

This section describes our implementation of the PIN cracking attack, through several optimization versions. We implemented all the versions in C with some embedded 80x86 assembly instructions. We used the Microsoft VC++ compiler on a PC running Microsoft Windows 98.

4.1 The Baseline

Before writing optimized versions of the code, we established two baseline implementations for comparison purposes, as follows.

4.1.1 The "as-is" version

This version is a non-optimized implementation of the attack, using C code only. The bias vectors (see Section 2.2.1) which are used during the SAFER+ key scheduling algorithm are calculated offline, and the substitution boxes **e** and **l** are implemented using two pre-calculated look-up tables.

4.1.2 The basic version

This version is identical to the "as-is" version, but with compiler optimizations to yield maximal speed[1].

4.2 Improved KSA & Expansion

Our first optimization technique focuses on the SAFER+ Key Scheduling Algorithm (KSA). We identified two effective optimizations in the KSA:

1. Caching the calculation result of the expansion operation in the E_{21} and E_{22} algorithms on the *BD_ADDR* of both peers. Since the input of *BD_ADDR* to E_{21} and E_{22} is nearly static (only two values of *BD_ADDR* are used during the PIN cracking attack), it is possible to perform the calculation of Expansion(*BD_ADDR*,6) (see Appendices B.1 and B.2) only once, and save the result for later use.

2. Enhancements of the implementation of the key scheduling algorithm. We found that the implementation of the byte-rotate operation (recall Section 2.2.1) using C code is expensive. Instead we used inline assembly code which employed the *ROL* instruction. Furthermore, we found that the modulo 17 operation used to extract specific bytes from a batch of 17 bytes during the key scheduling algorithm is very expensive. Instead, we used a pre-calculated look-up table.

4.3 PHT as lookup-table

In this version we used a large look-up table to implement the Pseudo Hadamard Transformation (PHT) operation, which is used 32 times during a single SAFER+ round. The look-up table is 65,536 entries long, since the transformation receives two bytes (2^8) and replaces them. The routine which implements the use of such a look-up table was written in pure assembly code. The look-up table was pre-calculated offline.

[1] Compile option /O2 yields maximal speed.

4.4 Algebraic Manipulation

Our most interesting and most effective optimization is the algebraic manipulation of the SAFER+ round. A key observation is that almost the entire SAFER+ round[2] can be implemented as a 16x16 matrix multiplying the vector of 16 input bytes (all operations modulo 256). This is possible since the operations used in the Armenian shuffles and Pseudo Hadamard Transformations are linear. By tracing back through the Shuffles and PHT boxes we computed the 16x16 matrix coefficients as follows:

$$\begin{pmatrix}
2 & 2 & 1 & 1 & 16 & 8 & 2 & 1 & 4 & 2 & 4 & 2 & 1 & 1 & 4 & 4 \\
1 & 1 & 1 & 1 & 8 & 4 & 2 & 1 & 2 & 1 & 4 & 2 & 1 & 1 & 2 & 2 \\
1 & 1 & 4 & 4 & 2 & 1 & 4 & 2 & 4 & 2 & 16 & 8 & 2 & 2 & 1 & 1 \\
1 & 1 & 2 & 2 & 2 & 1 & 2 & 1 & 4 & 2 & 8 & 4 & 1 & 1 & 1 & 1 \\
4 & 4 & 2 & 1 & 4 & 2 & 4 & 2 & 16 & 8 & 1 & 1 & 1 & 1 & 2 & 2 \\
2 & 2 & 2 & 1 & 2 & 1 & 4 & 2 & 8 & 4 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 4 & 2 & 4 & 2 & 16 & 8 & 2 & 1 & 2 & 2 & 4 & 4 & 1 & 1 \\
1 & 1 & 2 & 1 & 4 & 2 & 8 & 4 & 2 & 1 & 1 & 1 & 2 & 2 & 1 & 1 \\
2 & 1 & 16 & 8 & 1 & 1 & 2 & 2 & 1 & 1 & 4 & 4 & 4 & 2 & 4 & 2 \\
2 & 1 & 8 & 4 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 4 & 2 & 2 & 1 \\
4 & 2 & 4 & 2 & 4 & 4 & 1 & 1 & 2 & 2 & 1 & 1 & 16 & 8 & 2 & 1 \\
2 & 1 & 4 & 2 & 2 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 8 & 4 & 2 & 1 \\
4 & 2 & 2 & 2 & 1 & 1 & 4 & 4 & 1 & 1 & 4 & 2 & 2 & 1 & 16 & 8 \\
4 & 2 & 1 & 1 & 1 & 1 & 2 & 2 & 1 & 1 & 2 & 1 & 2 & 1 & 8 & 4 \\
16 & 8 & 1 & 1 & 2 & 2 & 1 & 1 & 4 & 4 & 2 & 1 & 4 & 2 & 4 & 2 \\
8 & 4 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 1 & 2 & 1 & 4 & 2
\end{pmatrix}$$

Our goal is to implement the multiplication (a 16 coefficients vector by a 16x16 matrix) faster than the traditional implementation of the Armenian shuffles and the PHT. A naive implementation of multiplying the vector with each column of the matrix would have taken 16 multiplication operations and 16 add operations for each column: 32 operations for each column, yielding 512 operations for the entire matrix (plus load and store operations). Such an implementation is slower than the traditional one: We found that each PHT box consists of 7 operations, the Armenian shuffle consist of 32 operations. This yields 320 operations for the traditional implementation (32 PHT boxes and 3 Armenian shuffles).

However, a careful examination of the above matrix shows that we can do much better. A much faster implementation is possible because the matrix has a great deal of structure, and because all the coefficients are powers of 2.

Observe that every pair of consecutive columns, starting with the two leftmost columns, are identical in half of their coefficients. All other coefficients in the left column are equal to twice the value of the coefficients in the right column. This structure is very useful, since the result of multiplication of half the column can be used in both columns. Furthermore, the product of the other coefficients can be calculated once and used for both columns, since they differ only by a factor of 2.

The fact that the coefficients are all powers of 2 is also helpful, since instead of using multiplication operations, the calculation is done using a shift left operation.

The next pseudo code depicts the calculation procedure for two columns. Note the saving in shift operations, done by arranging the add operation in an appropriate manner. The input vector is denoted by $X = (x_0, ..., x_{15})$, and we show the calculation of the outputs y_0 and y_1:

$$\begin{aligned}
h_1 &= x_1 + x_2 + x_3 + x_6 + x_7 + \\
&\quad 2(x_0 + x_5 + 2(x_4)) \\
h_2 &= x_8 + x_9 + x_{11} + \\
&\quad 2(x_{10} + x_{12} + x_{13} + 2(x_{15} + 2(x_{14}))) \\
y_1 &= h_1 + h_2 \\
y_0 &= y_1 + h_2
\end{aligned}$$

How fast is the new implementation?

This implementation consists of 5 shift left operations, 16 add operations, 2 load operations and 2 store operations. This yields 25 operations per 2 columns, **200** operations for the entire matrix multiplication: 30% fewer than needed in the normal implementation.

[2]Except the lookup tables **e** and **l** and the key addition steps.

PIN Length (digits)	Time (seconds)
4	0.063
5	0.75
6	7.609
7	76.127

Table 3: Summary of results obtained running the last version on a Pentium IV 3Ghz HT computer.

4.5 Results

This subsection presents the cracking time of the five versions. All the versions were run on an old Pentium III 450MHz Personal Computer. For each version we tried several PIN sizes, ranging from 4 to 7 decimal digits.

Figure 7 compares the results obtained from all five versions. The Y axis denotes the running time in seconds (logarithmic scale), and the X axis denotes the number of decimal digits the PIN contains.

The final version improves the cracking speed by a factor of 10, and brings the time to crack a 4-digit PIN down to 0.27 sec. To gain some insight on how the attack improves with stronger hardware, we also ran our best attack version on a Pentium IV 3Ghz HT. On this computer we were able to crack a 4-digit PIN in 63 msec

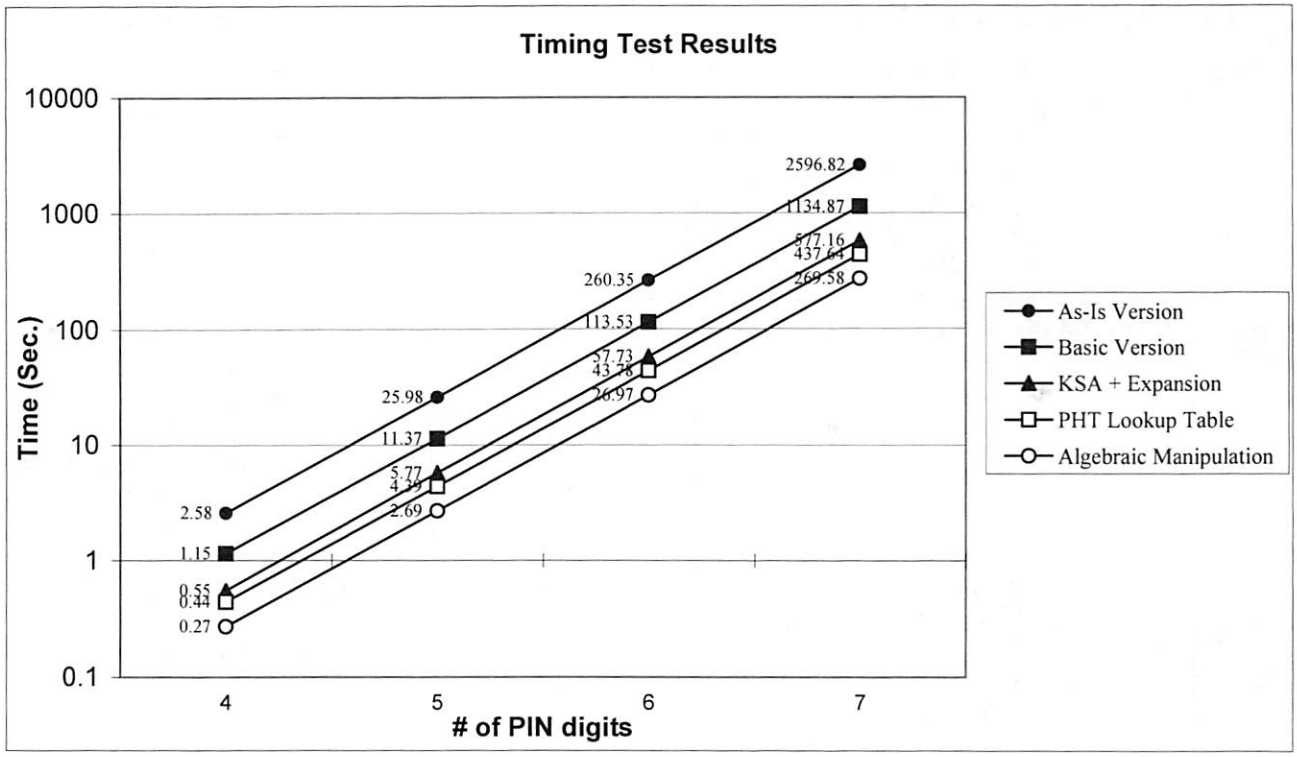


Figure 7: Timing results chart

(see Table 3) – 4 time faster than on the Pentium III. This makes the attack near-real-time.

5 The Re-Pairing attack

5.1 Background and motivation

This section describes an additional attack on Bluetooth devices that is useful when used in conjunction with the primary attack described in Section 3. Recall that the primary attack is only applicable if the attacker has eavesdropped on the entire process of pairing and authentication. This is a major limitation since the pairing process is rarely repeated. Once the link key K_{ab} is created, each Bluetooth device stores it for possible future communication with the peer device. If at a later point in time the device initiates communication with the same peer - the stored link key is used and the pairing process is skipped. Our second attack exploits the connection establishment protocol to force the communicating devices to repeat the pairing process. This allows the attacker to record all the messages and crack the PIN using the primary attack described in this paper.

5.2 Attack details

Assume that two Bluetooth devices that have already been paired before now intend to establish communication again. This means that they don't need to create the link key K_{ab} again, since they have already created and stored it before. They proceed directly to the *Authentication* phase (Recall Figure 3). We describe three different methods that can be used to force the devices to repeat the pairing process. The efficiency of each method depends on the implementation of the Bluetooth core in the device under attack. These methods appear in order of efficiency:

1. Since the devices skipped the pairing process and proceeded directly to the *Authentication* phase, the master device sends the slave an *AU_RAND* message, and expects the *SRES* message in return. Note that Bluetooth specifications allow a Bluetooth device to forget a link key. In such a case, the slave sends an *LMP_not_accepted* message in return, to let the master know it has forgotten the link key (see subsection 4.2.1.2 *"Claimant has no link key"* of Part C of Vol 2 of [Blu03]). Therefore, after the master device has sent the *AU_RAND* message to the slave, the attacker injects a *LMP_not_accepted* message toward the master. The master will be convinced that the slave has lost the link key and pairing

will be restarted (see subsection 5.1 *"AUTHENTICATION"* of Part C of Vol 3 of [Blu03]). Restarting the pairing procedure causes the master to discard the link key (see subsection 6.5 *"BONDING"* of Part C of Vol 3 of [Blu03]). This assures pairing must be done before devices can authenticate again.

2. At the beginning of the *Authentication* phase, the master device is supposed to send the *AU_RAND* to the slave. If **before** doing so, the attacker injects a *IN_RAND* message toward the slave, the slave device will be convinced the master has lost the link key and pairing is restarted. This will cause the connection establishment to restart.

3. During the *Authentication* phase, the master device sends the slave an *AU_RAND* message, and expects a *SRES* message in return. If, after the master has sent the *AU_RAND* message, an attacker injects a random *SRES* message toward the master, this will cause the *Authentication* phase to restart, and repeated attempts will be made (see subsection 5.1 *"REPEATED ATTEMPTS"* of Part H of Vol 2 of [Blu03]). At some point, after a certain number of failed authentication attempts, the master device is expected to declare that the authentication procedure has failed (implementation dependent) and initiate pairing (see subsection 5.1 *"AUTHENTICATION"* of Part C of Vol 3 of [Blu03]).

The three methods described above cause one of the devices to discard its link key. This assures the pairing process will occur during the next connection establishment, so the attacker will be able to eavesdrop on the entire process, and use the method described in Section 3 to crack the PIN.

In order to make the attack "online", the attacker can save all the messages transferred between the devices after the pairing is complete. After breaking the PIN (0.06-0.3 sec for a 4 digit PIN), the attacker can decode the saved messages, and continue to eavesdrop and decode the communication on the fly. Since Bluetooth supports a bit rate of 1 Megabit per second (see Part A of Vol 1 of [Blu03]), a 40KB buffer is more than enough for the common case of a 4 digit PIN.

Notes:

1. The Bluetooth specification does allow devices to forget link keys and to require repeating the pairing process. This fact makes the re-pairing attack applicable.

2. Re-Pairing is an active attack, that requires the attacker to inject a specific message at a precise point in the protocol. This most likely needs a custom Bluetooth device since off-the-shelf components will be unable to support such behavior.

3. If the slave device verifies that the message it receives is from the correct *BD_ADDR*, then the attack requires the injected message to have its source *BD_ADDR* "spoofed" - again requiring custom hardware.

4. If the attack is successful, the Bluetooth user will need to enter the PIN again - so a suspicious user may realize that his Bluetooth device is under attack and refuse to enter the PIN.

6 Countermeasures

This section details the countermeasures one should consider when using a Bluetooth device. These countermeasures will reduce the probability of being subjected to both attacks and the vulnerability to these attacks.

Since Bluetooth is a wireless technology, it is very difficult to avoid Bluetooth signals from leaking outside the desired boundaries. *Therefore, one should follow the recommendation in the Bluetooth standard and refrain from entering the PIN into the Bluetooth device for pairing as much as possible.* This reduces the risk of an attacker eavesdropping on the pairing process and finding the PIN used.

Most Bluetooth devices save the link key (K_{ab}) in non-volatile memory for future use. This way, when the same Bluetooth devices wish to communicate again, they use the stored link key. However, there is another mode of work, which requires entering the PIN into both devices every time they wish to communicate, even if they have already been paired before. *This mode gives a false sense of security!* Starting the pairing process every time increases the probability of an attacker eavesdropping on the messages transferred. We suggest not to use this mode of work.

Finally, the PIN length ranges from 8 to 128 bits. Most manufacturers use a 4 digit PIN and supply it with the device. Obviously, customers should demand the ability to use longer PINs.

7 Conclusion

This paper describes the implementation of an attack on the Bluetooth security mechanism. Our results show that using algebraic optimizations, the most common Bluetooth PIN can be cracked within less than 0.06-0.3 seconds. If two Bluetooth devices perform pairing in a hostile area, they are vulnerable to this attack.

References

[Arm02] Frederik Armknecht. A linearization attack on the Bluetooth key stream generator. Cryptology ePrint

Archive, report 2002/191, available from http://eprint.iacr.org/2002/191/, 2002.

[Blu03] Specification of the Bluetooth system, v.1.2. Core specification, available from http://www.bluetooth.org/spec, 2003.

[Blu04] Bluejackq. http://www.bluejackq.com/, 2004.

[FL01] Scott R. Fluhrer and Stefan Lucks. Analysis of the E_0 encryption system. In *Proc. 8th Workshop on Selected Areas in Cryptography, LNCS 2259*. Springer-Verlag, 2001.

[Flu02] Scott R. Fluhrer. Improved key recovery of level 1 of the Bluetooth encryption system. Cryptology ePrint Archive, report 2002/068, available from http://eprint.iacr.org/2002/068/, 2002.

[HN99] Miia Hermelin and Kaisa Nyberg. Correlation properties of the Bluetooth combiner generator. In *Information Security and Cryptology, LNCS 1787*, pages 17–29. Springer-Verlag, 1999.

[JW01] Markus Jakobsson and Susanne Wetzel. Security weaknesses in Bluetooth. In *Proc. RSA Security Conf. – Cryptographer's Track, LNCS 2020*, pages 176–191. Springer-Verlag, 2001.

[Kra02] Matthias Krause. BDD-based cryptanalysis of keystream generators. In L. Knudsen, editor, *Advances in Cryptology – EUROCRYPT'02, LNCS 1462*, pages 222–237. Springer-Verlag, 2002.

[Lau03] Adam Laurie. Serious flaws in Bluetooth security lead to disclosure of personal data. http://www.bluestumbler.org, 2003.

[LV04] Y. Lu and S. Vaudenay. Faster correlation attack on Bluetooth keystream generator E0. In *Advances in Cryptology – CRYPTO'04, LNCS 3152*, pages 407–425. Springer-Verlag, 2004.

[LW05] Ophir Levy and Avishai Wool. A uniform framework for cryptanalysis of the Bluetooth E_0 cipher. Cryptology ePrint Archive, Report 2005/107, 2005. http://eprint.iacr.org/2005/107.

[MKK98] J. L. Massey, G. H. Khachatrian, and M. K. Kuregian. SAFER+. In *Proc. First Advanced Encryption Standard Candidate Conference*. National Institute of Standards and Technology (NIST), 1998.

[Reh03] Gregory Rehm. 802.11b homebrew WiFi antenna shootout. http://www.turnpoint.net/wireless/has.html, 2003.

[Whi03] Ollie Whitehouse. War nibbling: Bluetooth insecurity. http://www.atstake.com/research/reports/acrobat/atstake_war_nibbling.pd%f, 2003.

[Whi04] Ollie Whitehouse. Bluetooth: Red fang, blue fang. CanSecWest/core04. Available from http://www.cansecwest.com/csw04/csw04-Whitehouse.pdf, April 2004. Vancouver, CA.

A Detailed specifications of SAFER+

A.1 SAFER+ Key Scheduling Algorithm

In continuation to section 2.2.1, the key scheduling algorithm uses 16 constant **Bias vectors**. The **Bias vectors**, denoted B_2 to B_{17}, are derived from the following equation:

$$B_c[i] = ((45^{(45^{17c+i+1} \bmod 257)} \bmod 257) \bmod 256), \quad \text{for } i = 0, .., 15.$$

One bias vector is used in each step, except for the first step. Note that the first step doesn't contain the cyclic rotate. Figure 8 describes the entire process of the key scheduling algorithm in SAFER+.

A.2 SAFER+ modified version

Besides using SAFER+ as is, Bluetooth uses a slightly modified version of SAFER+. This modified version is identical to the original SAFER+ implementation, only it also combines the input of SAFER+'s round 1 to the input of round 3: Some bytes are xored and some are added. This combination is done to make the modified version non-invertible. Figure 9 describes how the input of round 1 is combined with the input of round 3.

As stated before, all of the algorithms used during Bluetooth pairing and authentication process, use SAFER+ as is, or the modified version of SAFER+. In the remainder of this paper, SAFER+ is denoted as A_r, and the modified version of SAFER+ is denoted as A_r'. Next subsections describe how E_{22}, E_{21}, E_1 are implemented using SAFER+.

B SAFER+ Based Algorithms

B.1 Inner design of E_{22}

As described in subsection 2.1, E_{22} is used to generate the initialization key. The inputs used are:

1. a *BD_ADDR* (48 bits long).
2. the PIN code and its length L.
3. a 128 bit random number *IN_RAND*.

At first, the PIN and the *BD_ADDR* are combined to create a new word: if the PIN contains less than 16 bytes, some of the *BD_ADDR* bytes are appended to the PIN. If the PIN is less than 10 bytes long, all bytes of *BD_ADDR* shall be used. Let **PIN'** denote the new word created, and **L'** denote the number of bytes the new word contains. Now, if L' is less than 16, the new word is cyclic expanded till it contains 16 bytes. Let **PIN"** denote this

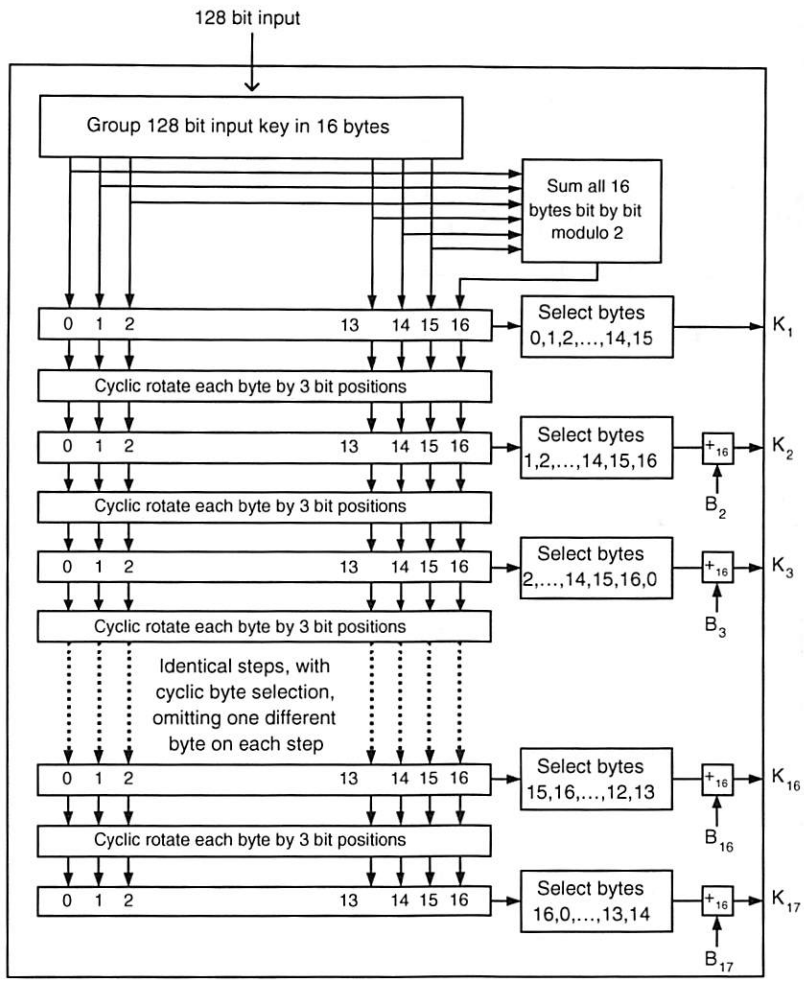


Figure 8: Inner design of SAFER+'s key scheduling algorithm

second new word. PIN" is used as the 128 bit input key of $A_r^{'}$. *IN_RAND* is used as the 128 bit input data, after xoring the most significant byte with L'. Figure 10 describes the inner design of E_{22}.

B.2 Inner design of E_{21}

As described in subsection 2.1, E_{21} is used to generate the link key. The inputs used are:

1. a *BD_ADDR* (48 bits long).
2. a 128 bit random number *LK_RAND*.

At first, the *BD_ADDR* is cyclic expanded to form a 128 bit word which is used as the input data of $A_r^{'}$. The key used for $A_r^{'}$ consists of the 128 bit random number *LK_RAND*, after xoring its most significant byte with 6 (result denoted ***LK_RAND'***). Figure 11 describes the inner design of E_{21}.

B.3 Inner design of E_1

As described in subsection 2.1, E_1 is used to perform mutual-authentication. The inputs used are:

1. A random word AU_RAND_A.
2. The link key K_{ab}.
3. a *BD_ADDR* (48 bits long).

The inner design of E_1 contains both A_r and $A_r^{'}$. The link key is used twice. Once, it is supplied as is for the key input of A_r. Later, it goes through a transformation denoted **Offset** and supplied as the key input of $A_r^{'}$. The "Offset" transformation consists of adding and xoring its bytes with some constants. For the full description of this transformation see page 778 of part H of Vol 2 of [Blu03].
As for the *BD_ADDR*, it is cyclic expanded to form a 128 bit word denoted *BD_ADDR'*. The inner design of E_1 is depicted in figure 12.

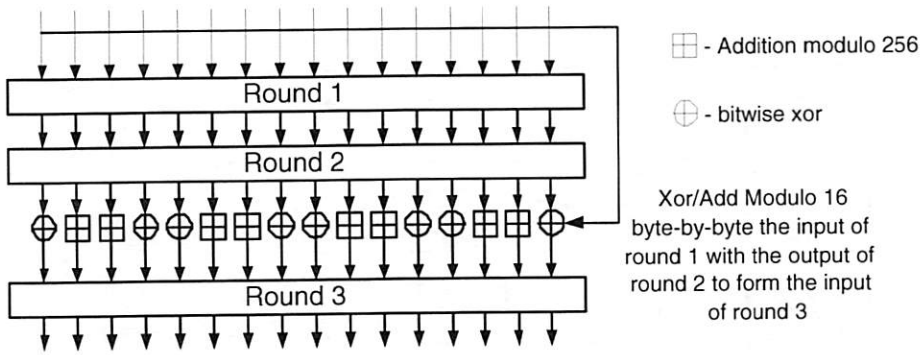


Figure 9: Combining input of round 1 with round 3 in SAFER+ modified version

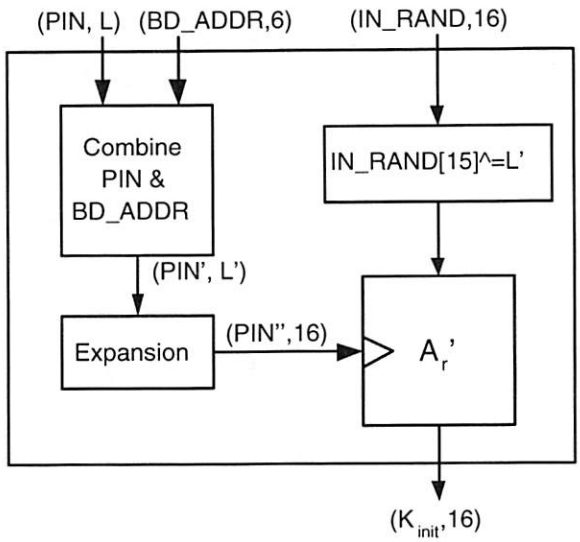


Figure 10: Inner design of E_{22}

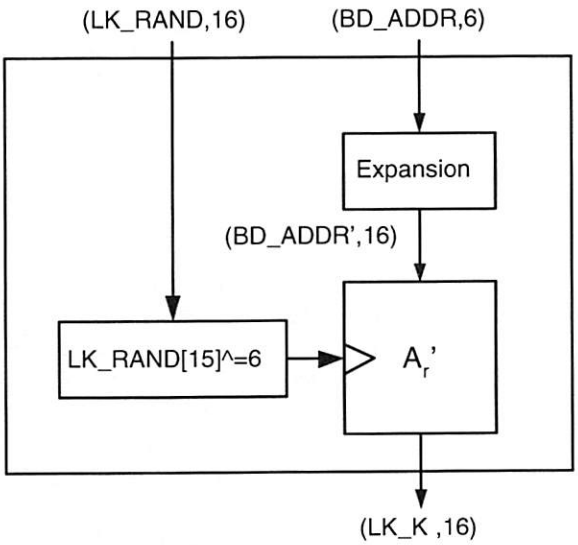


Figure 11: Inner design of E_{21}

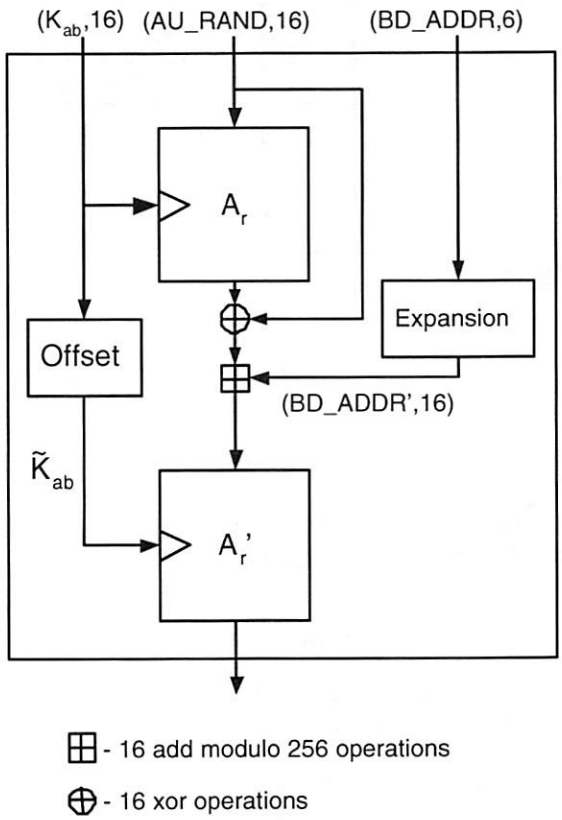


Figure 12: Inner design of E_1

Shake Them Up!

A movement-based pairing protocol for CPU-constrained devices

Claude Castelluccia
INRIA and University of California, Irvine
claude.castelluccia@inria.fr

Pars Mutaf
INRIA
pars.mutaf@inria.fr

Abstract

This paper presents a new pairing protocol that allows two CPU-constrained wireless devices Alice and Bob to establish a shared secret at a very low cost. To our knowledge, this is the first software pairing scheme that does not rely on expensive public-key cryptography, out-of-band channels (such as a keyboard or a display) or specific hardware, making it inexpensive and suitable for CPU-constrained devices such as sensors.

In the described protocol, Alice can send the secret bit 1 to Bob by broadcasting an (empty) packet with the source field set to Alice. Similarly, Alice can send the secret bit 0 to Bob by broadcasting an (empty) packet with the source field set to Bob. Only Bob can identify the real source of the packet (since it did not send it, the source is Alice), and can recover the secret bit (1 if the source is set to Alice or 0 otherwise). An eavesdropper cannot retrieve the secret bit since it cannot figure out whether the packet was actually sent by Alice or Bob. By randomly generating n such packets Alice and Bob can agree on an n-bit secret key.

Our scheme requires that the devices being paired, Alice and Bob, are shaken during the key exchange protocol. This is to guarantee that an eavesdropper cannot identify the packets sent by Alice from those sent by Bob using data from the RSSI (Received Signal Strength Indicator) registers available in commercial wireless cards. The proposed protocol works with off-the-shelf 802.11 wireless cards and is secure against eavesdropping attacks that use power analysis. It requires, however, some firmware changes to protect against attacks that attempt to identify the source of packets from their transmission frequency.

1 Introduction

The current trend in consumer electronics is to embed a short-range wireless transmitter and a microprocessor in almost everything. The main motivation is to facilitate communication and cooperation amongst wireless devices in order to reduce their size/cost and increase their functionality. In this context, each device can be seen as a peripheral of the others. For example, a user can use the display and the keyboard of a PDA to access his cellular phone or personal server [21]. Similarly, he can use a cellular phone or PDA to retrieve temperature data sensed by a local sensor [19].

The main security challenge is to securely associate the devices together. For example, when a device receives data from a sensor, it needs to make sure that the data is received from the sensor it has selected and not from an impostor. Furthermore, integrity and privacy are often very important too.

The process of securely associating two wireless devices is often referred to as *pairing*. This process allows two devices, communicating over a short-range radio, to exchange a secret key. This key can then be used to authenticate or encrypt subsequent communication. It is important to notice that the key exchanged in a pairing protocol does not need to be authenticated since the identities (often provided by certificates) do not matter in this context. A user who is pairing two devices together only needs assurance that a key was exchanged between the devices he/she has selected (for example, the two devices he/she is holding in his/her hands).

In summary, a pairing protocol is composed of two separate sub-protocols:

1. *Key exchange* sub-protocol: this protocol is run between the two wireless devices and results in a secret key shared between the two devices.

2. *Pairing validation* sub-protocol: this protocol is executed between the two wireless devices and the user. Its goal is to guarantee (with some large enough probability) to the user that a key was exchanged between the two devices he/she actually wished to pair.

Motivations and design constraints: The motivation of this work is to design a pairing protocol for CPU-constrained devices, such as sensors. Designing pairing protocols for such environment is very challenging because sensors have limited CPU and memory. Also, because of their low costs, most of them cannot rely on tamper resistant components. The consequence of the limited computing and storage capabilities is that modular arithmetic is difficult and therefore, asymmetric cryptography cannot be used. In particular, standard Diffie-Hellman (DH)[6] key exchange protocols are excluded. Even low exponent RSA[18] techniques that allow encryption cost to be minimized are prohibitive when sensors are involved. Our goal is to design a pairing protocol that meets these constraints.

More specifically, we aim at designing a protocol that does **not** use public key cryptography and does not rely on some preconfigured information. Furthermore, the designed protocol must not increase the complexity and the cost of the sensors by requiring additional hardware (a display, an I/O interface or an out-of-band channel, such as an infrared one). Finally, it should not require exotic wireless technologies, but instead work with current wireless networking standards such as 802.11 or 802.15.4 (an emerging Wireless PAN technology, designed for low power sensors). The proposed protocol must be secure against passive and active attacks. In other words, it must not allow active or passive attackers to learn the key exchanged between two paired devices. It must provide protection against Man-in-the-Middle (MitM) attacks that attempt to impersonate one or both of the devices during key agreement. It must also provide some protection against Denial-of-Service (DoS) attacks, i.e. prevent attackers from disrupting the pairing protocol and exhausting the devices' resources, such as their battery.

Contributions: We present a novel secure pairing technique based on a key agreement protocol that does not depend on CPU-intensive operations. Two CPU-constrained wireless devices A and B can establish a shared secret over an anonymous broadcast channel. An anonymous channel is a channel on which an eavesdropper can read the packets that are exchanged but is unable to identify the source. Using such a channel, A can send the secret bit 1 (resp. 0) to B by broadcasting an (empty) packet with the source field set to A (resp. B). Only B can identify the real source of the packet (since it did not send it, the source is A), and can recover the secret bit (1 if the source is set to A or 0 otherwise). An eavesdropper cannot retrieve the secret bit since it cannot figure out whether the packet was actually sent by A or B. By randomly generating n such packets A and B can agree on an n-bit secret key.

The protocol is secure if and only if the packets of A and B cannot be distinguished by an eavesdropper. On a wireless channel, this property is difficult to achieve through protocol design since an eavesdropper can measure the signal strength of each packet and may be able to determine the real source of each packet. Therefore, we propose that during key agreement, the user(s) be very close to each other and shake their devices (i.e. randomly turn and move one around the other) in order to randomize the reception power of their packets by a potential eavesdropper and make power analysis very difficult.

Organization: The paper is structured as follows: Section 2 presents the related work. Section 3 presents the basic ideas of our scheme. Section 4 describes our proposal in detail. Section 5 presents experimental results and analysis. Section 6 presents a discussion. Finally, Section 7 concludes the paper.

2 Related work

2.1 Secure pairing

The problem of secure pairing of wireless devices has been tackled by several researchers. The proposed *key-exchange* solutions can be classified into the four main categories described in this section.

All of the approaches that we review below require some additional mechanism for *pairing validation* (except the last two ones). The only solution proposed in the literature so far is to provide the user with some evidence that both devices computed the same secret key. For example, the devices can both display a hash of the secret key [7]. These solutions are not always practical since they require devices with a display and/or a keyboard.

2.1.1 Public-key cryptography based solutions

These solutions rely on public-key based key exchange protocols such as Diffie-Hellman or RSA [7, 10, 11]. In Diffie-Hellman based schemes, devices exchange their Diffie-Hellman components and derive a key from them. In RSA-based schemes, one of the devices selects a secret key and encrypts it under the other device's public key. The main problem of these solutions is performance. They require that devices perform CPU-intensive operations such as exponentiation, which are prohibitive for CPU-constrained devices.

2.1.2 PIN-based schemes

In Bluetooth, two wireless devices derive a shared key from a public random value, the addresses of each device and a secret PIN number. The PIN number is provided to each device by the user via an out-of-band channel, such

Clearly, in both cases, having small and lightweight devices (e.g. sensors or small PDAs) will reduce the user burden for key agreement. The user can take one device in each of his hands and randomly move them one around the other according to the horizontal and vertical axes. If the devices are very small, he can take both of them in one hand and shake his arm, like he would do with an orange juice bottle.

The security of the proposed scheme depends on the quality of the movement. Users of our scheme should be aware that it is their responsibility and in their best interest to move the devices properly during the pairing phase. Movement-based operations or "protocols" are quite frequent and accepted in our everyday lives. For example, orange juice or shaving cream bottles are universally shaken prior to use. This is now a well-known and quite a natural protocol. Furthermore it is commonly accepted that it is the responsibility and in the interest of the consumer to perform this shaking operation properly.

4.3 Protection against MitM and DoS attacks

4.3.1 Protection against MitM attacks

Defeating a MitM (Man-in-the-Middle) attack requires assurance for a terminal A that a secret key is really being exchanged with the intended terminal B and not an impostor's device. This is the goal of the *Pairing Validation* protocol, as described in Section 1.

In our case, this problem is reduced to the following issue: "how do the two devices reliably determine each other's address in the presence of many other devices?" As explained in Section 2.2, the smart-its friends scheme solves this problem. However this solution may be considered costly and impractical since it requires extra hardware, namely an accelerometer. Certificates could also be used for authenticating the START messages. However, our goal in this paper is to avoid CPU-expensive operations such as exponentiations or signature verifications.

In our case, $proximity$ (and hence high signal level of START messages) is used for authentication. In "Shake Them Up!", the pairing protocol is triggered by the user by, for example, pushing a button on the devices. At this point, the devices will start to generate START messages at a specified rate. The user will bring the two devices near to each other (possibly resulting in antenna contact). The devices will receive each other's START message with a high signal level, starting the "Shake Them Up!" procedure. Experience with 802.11 cards has shown that a very high signal level can be obtained when two cards touch each other, and a distance about 1 or 2 cm quickly results in a much lower signal level. Using a signal level threshold e.g. 0 or 1 dBm, this device association protocol can be implemented. In our case, the higher the specified rate of START messages, the faster the devices can detect each other with high signal level (when the user finds the correct positioning). Let Q be the period of broadcast START messages. While initiating the pairing process, if the user missed a START message (i.e. it was received at a lower signal level than the specified threshold), the user must wait another Q time units. Consequently, Q must be low, e.g. for example 1 second, to allow the user to easily and quickly start the pairing process. Once the two devices obtain each other's address correctly, MitM attacks will be impossible.

A distant impostor may attempt to foil this technique by increasing its transmission range. However this attack is easily detectable since the victim will receive several START messages at a rate higher than Q.

One of the devices, say device B, may be down and an attacker may profit from this situation to impersonate B. This attack can be prevented if each device has a status LED which indicates whether it is active or down. Even a sensor device (with no display) is likely to have such a LED. Furthermore, if the devices A and B are shaken together (i.e. the user holds the two devices in one hand and shakes his hand), A should receive the messages coming from B with constant signal power. Most likely, the signal power of the different messages sent by the attacker will be different (since the attacker is not shaken together with A and therefore does not follow the same mobility pattern). In this scenario, this attack can easily be detected by A.

4.3.2 Protection against DoS attacks

We can differentiate between two kinds of DoS (Denial-of-Service) attacks on a key agreement protocol. In the first one an attacker may exploit the key agreement protocol to force a victim to perform computationally expensive operations, with the goal of draining its battery or preventing it from performing useful work. Unlike public-key cryptography-based schemes, our protocol is not based on CPU-intensive operations and therefore immune against such DoS attacks. Another DoS attack may consist of sabotaging the key agreement, i.e. making it impossible for the two parties to agree on a same secret key.

In our basic scheme, it is easy for an attacker to insert a bogus packet with source address A and destination address B (or vice versa) and perform what we call a *key poisoning* attack. Such a packet inserted by a third party would generate different secret bits at the terminals A and B. The attacker can insert an arbitrary number of bogus packets, and make it impossible for A and B to agree on a secret key.

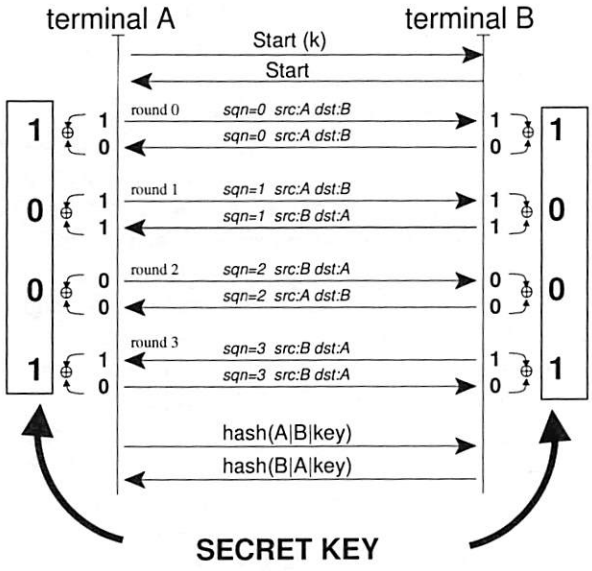


Figure 2: A protocol that resists what we call a *key poisoning DoS attack*. When this protocol is used for key agreement, an active attacker cannot poison (i.e. corrupt) the secret key by inserting bogus messages with the addresses of A and B (see text for details). This protection is obtained at the cost of two messages per secret bit.

The protocol depicted in Figure 2 defeats the key poisoning attack. In this protocol, each secret bit is constructed using one packet from A and another from B, i.e. both terminals contribute to the construction of each secret bit. Each secret bit is given a sequence number (which also corresponds to the round number). In order to generate the secret bit i the two terminals generate a packet with correct or flipped address positions, in random order (i.e. the probability that the first packet i will be transmitted by A is 0.5). The outcome of the two packets i are combined by taking their sum (mod 2), or exclusive OR. The result is the secret bit i.

In order to alterate the secret bit i, an attacking node can insert x packets with the same sequence number. In this case both sides will record $x + 2$ bits with the same sequence number, but only two of them will be the same at both sides. Let $\{a_1, ..., a_{x+2}\}$ and $\{b_1, ..., b_{x+2}\}$ the set of bits (with the same sequence number) that the terminals A and B have recorded. Then we have

$$a_1 \oplus \cdots \oplus a_{x+2} = b_1 \oplus \cdots \oplus b_{x+2}$$

if x is even, and

$$b_1 \oplus \cdots \oplus a_{x+2} \neq b_1 \oplus \cdots \oplus b_{x+2}$$

if x is odd.

Key poisoning can be defeated by taking the sum (mod 2) of the bits with the sequence number i, as usual (except that in normal operation $x = 0$). Note that if the attacker inserted an odd number x of packets, the terminal A must invert the resulting secret bit i. This protocol requires twice as many messages as the basic protocol. However, this protocol is only necessary when the user believes that his devices are under DoS attack. Otherwise, the basic scheme should be used.

This protocol fails, however, when A and B do not receive the same messages. This might happen when some of the messages are lost. We therefore suggest that A and B append to each of their messages a hash of all the previous messages they have seen since the beginning of the protocol. As a result, if one or several messages are lost, A and B can detect it immediately (instead of waiting until the end of the protocol and comparing a hash of the derived keys).

5 Experimentations and analysis

In this section, we analyze, by experimentations, the security of the proposed pairing scheme. More specifically, we show that signal power analysis cannot be used by an attacker to retrieve the key exchanged between two devices that use our pairing protocol. We also evaluate the energy cost of our protocol and show that although it requires several messages, it is much more energy-efficient than a Diffie-Hellman based pairing protocol.

5.1 Setup and methodology

We built a testbed with lightweight laptops equipped with a PCMCIA Lucent IEEE 802.11 Wavelan card operating at 2.457GHz (802.11 channel 10) and 2Mbps bit rate and in ad hoc mode. Our cards support RSSI (Received Signal Strength Indicator) and allow us to visualize and evaluate the power of received packets. Our signal level cryptanalyzer is built upon Linux wireless tools[3], and in particular *iwspy* that allows to get per node link quality. The *iwspy* command takes as argument a MAC address M, and outputs the received signal and noise levels of packets whose source address is M.

Figure 3 depicts a typical measurement that can be carried out by any user using the *iwpsy* tool and a simple sampling script. Note that *iwspy* also outputs the noise level which is about -96dBm in our environment.

The received signal strength depends on at least 4 distinct factors:

1. Transmission power: Packets are transmitted at our cards' default value which is 15 dBm[4].
2. Distance between the source and the signal level analyzer.
3. The relative angle between the source and the signal level analyzer: the cards that we use are not

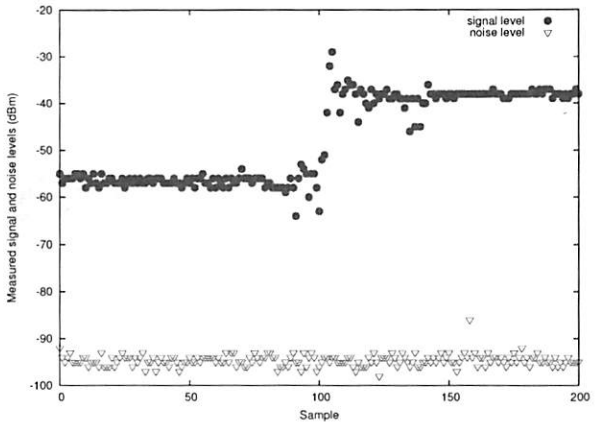


Figure 3: A typical *iwspy* output. In this example, the "iwspied" terminal is stationary while the first ~100 samples are taken and then it moves to a location that is closer to the signal level analyzer machine. The distance between the two cards is easily determined by signal strength analysis.

omni-directional and the received signal level depends heavily on the relative angle of the two cards.

4. Relative position of the terminals (or, cards): when the two terminals are very close, they may become obstacles for each other. For example, terminal A may be in front of terminal B. In this case, packets from A are received at a higher signal level (even if the above factors had no impact on the signal level difference).

During each experiment, referred as 'scenario', Eve (eavesdropper, or cryptanalyzer) measures the signal strength of the packets sent by Alice (terminal A) and Bob (terminal B) during key agreement. Many different experiments were carried out. We only provide the most representative ones that we consider generic and applicable to almost all situations because they perfectly reflect the points (2) and (3) listed above. Our first scenario, denoted *scenario1*, is illustrated in Figure 4-a. In this scenario Alice and Bob are close to each other (within 0.5 meter) and make two kinds of movements in order to equalize their average signal strength captured by Eve:

- We use commodity wireless Ethernet cards and they are not omnidirectional. Thus, in order to confuse Eve, Alice and Bob must turn their laptop in randomly changing directions (at a reasonable speed). The process requires reasonable effort from the user and takes around 16 seconds for agreeing upon an 80-bit secret key. The details of movement speed and its effect on security will be discussed later.
- Alice and Bob move their devices one around another with a reasonable effort, i.e randomly and at a moderate speed. This helps Alice and Bob to hide their relative distance between their cards and a potential eavesdropper that may be located anywhere.

In scenario1, we consider a pessimistic passive attack where Eve's wireless Ethernet card (the white arrow) is directly oriented to Alice and Bob and situated only 2.2 meters away. This allows Eve to make relatively accurate signal level measurements. In practice, Alice and Bob would probably notice the presence of a third person during key agreement, and look for another place where eavesdroppers cannot approach them. However, in some situations such countermeasures may not be practical. This scenario attempts to capture the cases where the presence of a third person cannot be avoided. Note also that an eavesdropper may have installed hidden signal level cryptanalyzers at strategical points. Thus, the absence of a third person, does not necessarily imply a secure environment. For example, the eavesdropper might be behind a thin wall or partition (e.g. a cubicle wall), and not visible to Alice and Bob. In this scenario Alice and Bob respect the key agreement requirements, hence the key will be secure as we will show below.

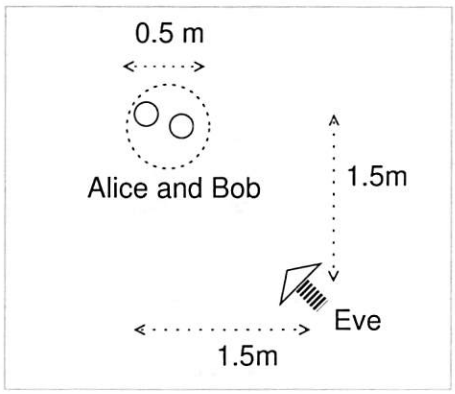
In scenario2 (Figure 4-b), we demonstrate an inappropriate usage of our protocol that we would like to disadvise. In this scenario Alice and Bob are not close to each other. They both move their laptop randomly in every possible direction, but they are always far from each other and their location does not change during key agreement. Eve profits from the distance between Alice and Bob, and directs her card to Alice. Consequently, Alice's packets are received at a higher signal level than that of Bob, rendering the secret key weak.

The scenario3 (Figure 4-c) is even less secure and firmly disadvised. Alice is 4 times closer to Eve than Bob. Eve is located between the two terminals and profits from the situation by directing her wireless Ethernet card to Alice. Although Alice and Bob's cards are perfectly turned in random directions, Eve can easily differentiate between their packets. As a result, the key is extremely weak.

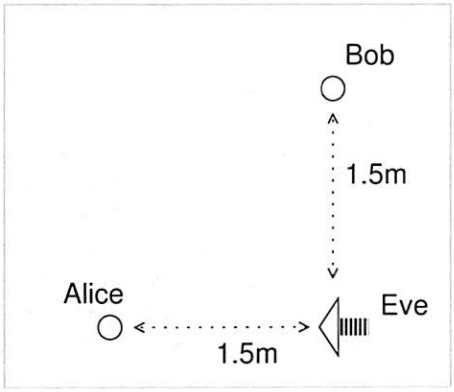
5.2 Signal power analysis

The signal level cryptanalysis results for the above three scenarii are shown in Figure 5.

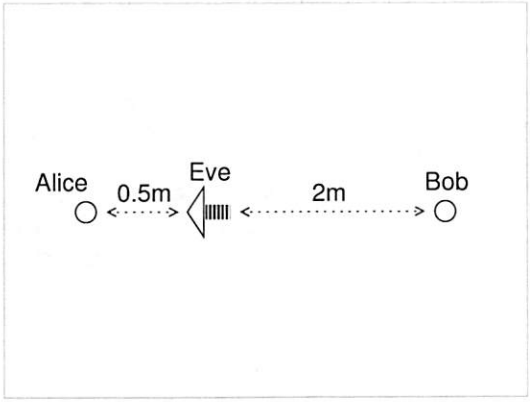
In scenario1, we observe that Alice and Bob's packets are mixed and not easily distinguishable by Eve (the reader may imagine that Eve's vision has only one color regardless of the sender's ID). The only information available to Eve will be the absolute value of signal level difference between two packets captured during



(a) scenario1

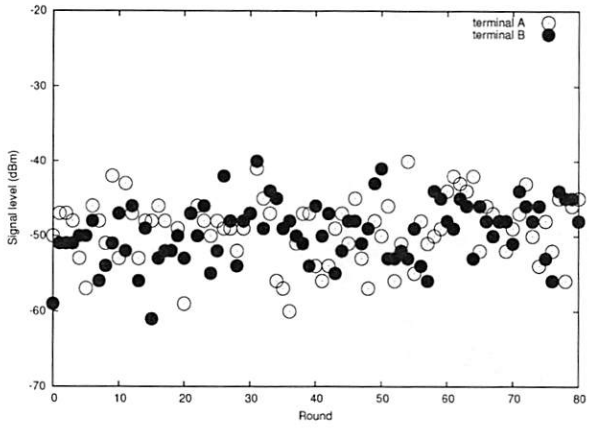


(b) scenario2

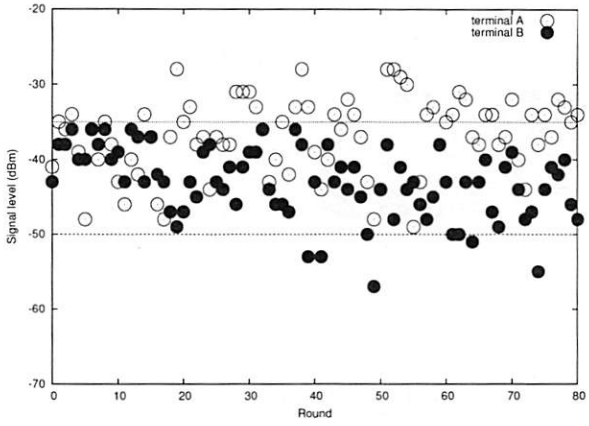


(c) scenario3

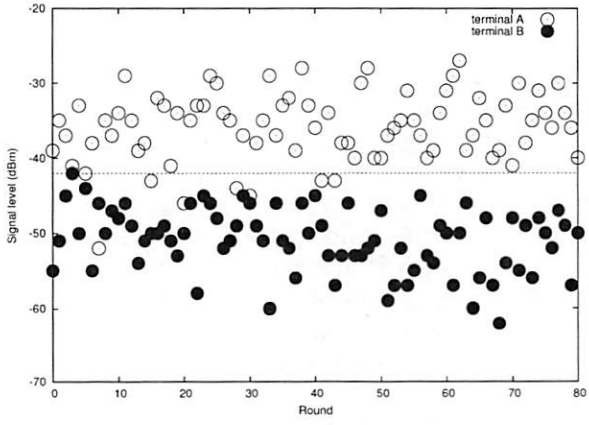
Figure 4: Experimentation scenarii.



(a) scenario1



(b) scenario2



(c) scenario3

Figure 5: Received signal level of terminal A (Alice) and terminal B (Bob) during key agreement. The reader may imagine that Eve's vision has only one color (i.e. all packets are black).

each round (1 packet from Alice, 1 packet from Bob). In Figure 6 we provide a frequency diagram of these signal level differences. It should be noted that, when plotting these histograms we have profited from additional information that is not available to Eve: the sign of the observed differences (i.e. (+) when Alice's packet is received with greater signal level than that of Bob, and (-) otherwise). These histograms were plotted using 1000 samples (i.e. rounds) in order to provide accurate results that correspond to the average case (for a given scenario). For our data collection purposes, Alice and Bob performed the required laptop movements for a much longer time than needed in practice: $\sim$ 3.5 minutes for each experiment (in our experiments Alice and Bob generated 0.2 packets per second, as we will explain later). The histogram that corresponds to scenario1 is centered on $\sim$0 and roughly symmetric. Consequently, we conjecture that Alice and Bob's packets cannot be distinguished using signal strength analysis.

In practice, the results look satisfactory in scenario2. There is no well defined technique (at least to our knowledge at time of writing) that will allow to clearly distinguish Alice's and Bob's packets. Note for example that, above the line -50dBm all packets are Alice's packets. Similarly, below the line -35dBm, we have only Bob's packets. However, unlike the reader, this information is not provided to Eve.

On the other hand, the resulting key is clearly insecure in theory. As shown in Figure 5-b the signal level differences are important: the histogram is centered at 7.34 dBm. Although it is unknown to the attacker, this difference is considerable (making us uncomfortable) and reflects very well the fact that Eve's wireless Ethernet card is directed to Alice. Thus, we base our conclusion on the theoretical security of the protocol and disadvise the type of scenario where Alice and Bob are 'not' close to each other. The security of the secret key in this scenario could be improved by adding more rounds, however this would lead to a very inefficient key agreement. We recommend that Alice and Bob be as close to each other as possible (in addition to turning their devices in random directions).

In the final scenario, scenario3, the situation is clearly worse. The resulting key is not only 'theoretically' breakable as shown by the frequency diagram (centered at 14.92 dBm), but also breakable in practice. Figure 5-c reveals what we call a "break point" which is situated around -41dBm. There is a visible gap at that point where Alice and Bob's packets are clearly separated.

5.3 Energy consumption considerations

In this section, we compare the power consumption of our scheme with the power consumption of a Diffie-Hellman based pairing. For the purpose of this comparison, we assume that the two devices being paired are sensors using TinyOS and that the size of the generated shared key is 72 bits.

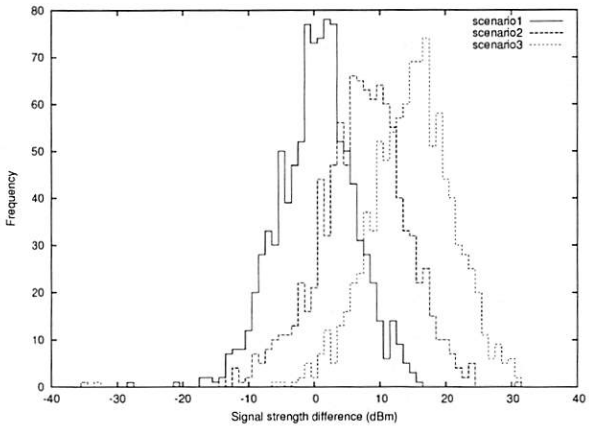


Figure 6: Frequency diagrams for signal level difference. In scenario1, the spatial indistinguishability requirement is satisfied. The histogram is centered on zero and symmetric. Thus, in this paper it is conjectured that an eavesdropper cannot distinguish the source of the packets regarding signal level difference (in scenario1).

With our scheme, each device must receive and send 36 packets. Considering that a TinyOS packet that has a header size of 7 bytes [13], each device must send and receive 2016 bits ($36 \times 8 \times 7$) (as explained in Section 4, in our basic protocol, packets do not have to contain any payload). However transmitting one bit consumes about as much power as executing 800-1000 instructions [9, 13]. Receiving one bit consumes about half as much power as sending one bit. As a result, our protocol consumes as much energy as the execution of about 2.72×10^6 instructions ($2016 \times 900 + 2016 \times 450$).

In comparison, with a Diffie-Hellman based pairing protocol, each device needs to exchange their Diffie-Hellman public component (i.e. g^x, where x is the device's private key). A security equivalent to 72 bits requires to select a modulus of 1024 bits and an exponent of 130 bits [14]. As a result, the device's Diffie-Hellman public component is 1024-bit long. Since the maximum number of payload bits in a TinyOS packet is 232, each device must send (and receive) 5 packets. Therefore, the total number of bits sent and received is 1304 bits: 4 packets containing 232 bits and 1 packet containing 96 bits. This consumes as much energy as the execution of 1.76×10^6 instructions ($1304 \times 900 + 1304 \times 450$). Upon reception of the other party's public component, each device has to exponentiate it with its Diffie-Hellman private key. Exponentiating using the Montgomery algorithm re-

quires $3 \times l \times (l+1) \times (t+1)$ single-precision multiplications, where l is the size of the modulus and t the size of the exponent [16]. With $l = 1024$ and $t = 130$, each device must perform 4.12×10^8 single-precision multiplications. In conclusion, the total power consumed by each device is therefore equivalent to the power consumed by the execution of $1.76 \times 10^6 + 4.12 \times 10^8$ instructions. This cost is about *100 times* larger than the cost of our scheme.

The bandwidth cost of the Diffie-Hellman based solution could be significantly decreased with Elliptic Curve Cryptography. In fact, the security of a 1024-bit Diffie-Hellman key exchange is equivalent to the security of a 135-bit Elliptic Curve Diffie-Hellman (EC-DH) key exchange [14]. Therefore only one packet would be necessary to be exchanged by the two devices. This would reduce the energy cost due to communication by 5. However, as shown in [5], EC-DH key derivation cost is even more expensive than regular DH key derivation. Therefore the total energy cost would still be much higher than the cost of our scheme.

6 Discussion

In this section, we present a discussion of more sophisticated attacks that the "Shake Them Up!" strategy may face.

6.1 RF analysis attacks on reference clocks

How secure is our protocol against a well equipped attacker? An attacker may use more sophisticated equipment, and conduct much more rigorous cryptanalysis. Using an RF test equipment, for example, it is possible to snoop the packets and record their signal shape. By studying the signal shape of each packet, additional information may be discovered and help distinguish one of the device's packets from the other. Although the signal amplitude should not reveal anything more than an RSSI measurement, signal-frequency may reveal a drift between the participants' reference clock implementation.

By current practice, a quartz crystal or crystal clock oscillator stabilizes the carrier and baseband frequencies in an RF transceiver. In order to ensure frequency lock between two devices, and avoid serious phase noise, a tight-stability reference clock is necessary. Nevertheless, a reference clock implementation is never perfect. A random clock drift is generally unavoidable, due to practical difficulties found at the hardware level. Typically, an error of up to ±25ppm (parts per million) is tolerated. This tolerance includes the initial calibration tolerance at 25°C, frequency changes over operating temperature, power and load fluctuations, and aging[17]. For example, at 2.4GHz carrier frequency, a frequency offset of up to $\frac{2.4\times10^9\times2\times25}{10^6}$ =120kHz would be tolerated. [4] reports 250kHz of central frequency accuracy tolerance. The main reason for clock drift is aging. I.e. the clock drift is mostly stable in short-term (except in case of shock, or abrupt temperature changes), but logarithmically increases in time[20, 1]. A clock drift from ±1 to ±5 ppm/year can be incurred depending on the crystal used [2].

Consequently, during "Shake Them Up!", device A may always transmit at a central frequency f_A while the device B transmits at f_B, where $f_A = f_B + \epsilon$. A third party equipped with an RF analyzer can then retrieve the secret key by correlating the packets with the same central frequency. A frequency shift of several kHz is unfortunately too large to be compensated with the *Doppler effect* made by separately shaking the devices. A shaking speed of ±10m/s would only make a Doppler effect between ±50Hz (at 2.4GHz) which probably cannot counterbalance the error ϵ.

A possible defense against this attack consists of adding a deliberate and random frequency offset so that f_A and f_B span over similar frequency ranges. This solution however requires firmware changes, making it a longer-term solution. Let the frequency offset tolerance be $\pm\delta$ and $f_A < f_B$ (i.e. f_A and f_B are within the range $[f - \delta; f + \delta]$) and both devices add a deliberate random frequency shift t to each packet. The devices will have a frequency range between $[f_A - t; f_A + t]$ and $[f_B - t; f_B + t]$ respectively. Assuming that an eavesdropper, Eve, knows f_A and f_B, then she can deduce that the packets transmitted with a frequency larger than $f_A + t$ originate from B, and the packets transmitted with a frequency smaller than $f_B - t$ originate from A.

However, the packets received within range $[f_B - t; f_A + t]$ are emitted by A or B with the same probability. This is illustrated in Figure 7. Let k be the number of packets emitted by each device. It can be shown that, $\frac{k}{2t} \times (f_A - f_B + 2t)$ packets of A (and B) will have a central frequency between $[f_B - t; f_A + t]$. If we wish to use an 80-bit secret key, at least 40 packets of A and 40 packets of B must be in this frequency range. This condition is satisfied if: $\frac{k}{2t} \times (f_A - f_B + 2t) > 40$, i.e.

$$k > \frac{40 \times 2t}{f_A - f_B + 2t}$$

where $t > \delta$ (recall that 2δ is the maximum frequency shift between two devices). In the worse scenario, $f_A = f - \delta$ and $f_B = f + \delta$, i.e. $f_A - f_B = -2 \times \delta$ and $k > 40 \times t/(t - \delta)$. If t is set to $2 \times \delta$, then k can be set to 80. To summarize, by setting t to $2 \times \delta$ and k to 80 (instead of 40 in the basic scheme), the generated secret is secure for any value of f_A and f_B in $[f - \delta; f + \delta]$.

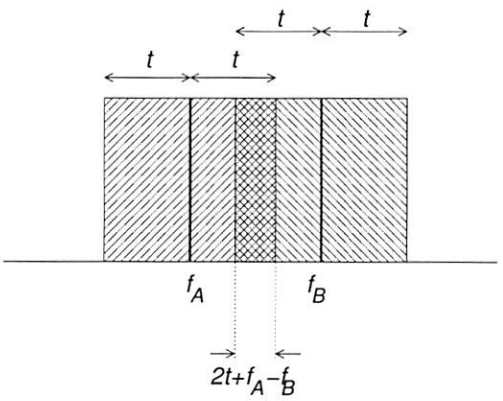


Figure 7: A technique for hiding the clock drift between two devices. In the region where the central radio frequencies of each device overlap, the packets of A and B are indistinguishable.

6.2 Camera-assisted packet captures

Another type of attack that "Shake Them Up!" may be exposed to is a video camera assisted attack. Using a signal level analyzer that is synchronized with a video camera, an attacker may correlate different device locations and their respective signal power during the shaking process. Users may employ different strategies against this threat such as hiding the sensor devices with their hand (provided that the devices are small).

Hiding the devices has also the additional benefits that it protects our scheme against eavesdroppers with unidirectional antennas. Since the location of the sensors are hidden, an eavesdropper cannot aim an antenna at one of the sensors for identifying its packets.

7 Conclusion

In this paper we presented a novel secure pairing scheme for CPU-constrained devices without a need for special hardware or interfaces. Using an existing communication channel such as 802.11 or 802.15.4, two devices that are close to each other can agree on a secret key using an algorithm that does not depend on CPU-intensive operations. On the other hand, user assistance is required for *shaking* the devices during key agreement in order to preserve key secrecy.

One alternative could consist of randomly varying the signal level (in software) during key agreement. However, this solution is not secure because an eavesdropper may aim an unidirectional antenna at one device, identify its packets and therefore retrieve the secret key. Furthermore we have discovered by experimentations that if the two devices are not shaken, one of them can mask the signal of the other one and attenuate its transmission power significantly. Consequently, the packets from each device are received at a different signal level, and the secret key can easily be retrieved. Shaking solves all these problems. It seems to be the only solution that can address all kinds of RSSI-based signal level analysis threats that our key agreement protocol may face. The proposed protocol works with off-the-shelf 802.11 wireless cards and is secure against eavesdropping attacks that use power analysis. It requires, however, some firmware changes to protect against attacks that attempt to identify the source of packets from their transmission frequency.

One limitation of our scheme is that it is specific to random media access technologies. For example, it is not suitable for TDMA-based protocols and, therefore, cannot be used with Bluetooth devices. Our scheme requires CSMA-based systems, such as 802.11 or 802.15.4 (an emerging Wireless PAN technology, designed for low power sensors). Another noticeable limitation is that it requires that the transmission power of both devices be similar. This was the case with the 802.11 devices that we used for our experimentations. However, for some wireless technologies, a power control protocol might be required to adjust the transmission power accordingly.

Objects with microprocessors and wireless transceivers surround us. Today's users are more and more technology and security-aware. Almost all users today learned that a system access password should contain non-alphanumeric characters. We have learned (or are forced to learn) how to handle computer viruses. Technology and information security have become part of our everyday lives. Thus, we believe that future users can also learn that *two small devices must be shaken well before secure use*. This is actually a very common protocol that we already execute everyday. For example, orange juice or shaving cream bottles are universally shaken/moved before usage. This is now a well-known and quite a natural "protocol". Furthermore it is commonly accepted that it is the responsibility and in the interest of the consumers to perform this shaking operation properly. Similarly, in our case by shaking the devices well, the user can make sure that the two devices are paired securely.

Acknowledgement

We would like thank Roy Want, Gene Tsudik and the anonymous reviewers for their excellent remarks that helped us improve this paper.

References

[1] Fundamentals of Quartz Oscillators. HP Application Note 200-2.

[2] http://www.telluriantech.com. Specialty Crystals, Quartz Crystals.

[3] ALPERN, B., AND SCHNEIDER, F. Key exchange using "Keyless Cryptography". *Information processing letters 16*, 2 (February 1983), 79–82.

[4] CHAYAT, N. 802.11a PHY Overview. Slides available at: http://www.nwest.nist.gov/mtg3/papers/chayat.pdf.

[5] DAI, W. Speed benchmarks for various ciphers and hash functions. URL:http://www.eskimo.com/~weidai/.

[6] DIFFIE, W., AND HELLMAN, M. New directions in cryptography. *IEEE Transactions on Information Theory IT-22*, 6 (1976), 644–654.

[7] GEHRMANN, C., AND NYBERG, K. Enhancements to bluetooth baseband security. In *Nordsec'01* (Kopenhagen, Denmark, November 2001).

[8] GOLDWASSER, S., AND BELLARE, M. Lectures notes in cryptography. URL:http://www.cs.ucsd.edu/users /mihir/papers/gb.html.

[9] HILL, J., SZEWCZYK, R., WOO, A., HOLLAR, S., CULLER, D. E., AND PISTER, K. S. J. System architecture directions for networked sensors. In *Architectural Support for Programming Languages and Operating Systems* (2000), pp. 93–104.

[10] HOEPMAN, J.-H. Ephemeral pairing in anonymous networks. Available at: http://www.cs.kun.nl/~jhh/publications/anon-pairing.pdf.

[11] HOEPMAN, J.-H. The ephemeral pairing problem. In *8th Int. Conf. Financial Cryptography* (Key West, Florida, February 9-12 2004), pp. 212–226.

[12] HOLMQUIST ET AL, L. A. Smart-Its Friends: A Technique for Users to Easily Establish Connections between Smart Artefacts. In *Ubicomp 2001* (Atlanta, Georgia, September 30, October 2 2001).

[13] KARLOF, C., SASTRY, N., AND WAGNER, D. Tinysec: A link layer security architecture for wireless sensor networks. In *Second ACM Conference on Embedded Networked Sensor Systems (SenSys 2004)* (November 2004).

[14] LENSTRA, A. K., AND VERHEUL, E. R. Selecting cryptographic key sizes. *Journal of Cryptology: the journal of the International Association for Cryptologic Research 14*, 4 (2001), 255–293.

[15] LESTER, J., HANNAFORD, B., AND G., B. "Are You with Me? - Using Accelerometers to Determine If Two Devices Are Carried by the Same Person". In *Pervasive 2004* (Vienna, Austria, April 21-23 2004).

[16] MENEZES, A. J., VAN OORSCHOT, P. C., AND VANSTONE, S. A. *Handbook of applied cryptography*. CRC Press series on discrete mathematics and its applications. 1997. ISBN 0-8493-8523-7.

[17] OGILVIE, B. Clock Solutions for WiFi (IEEE 802.11). Saronix(tm) application note, 2003.

[18] RIVEST, R., SHAMIR, A., AND ADLEMAN, L. A method for obtaining digital signatures and public-key cryptosystems. *Coomunications of the ACM 21* (1978), 120–126.

[19] STAJANO, F., AND ANDERSON, R. The resurrecting duckling: Security issues for ad-hoc wireless networks. In *Proceedings of the 7th International Workshop on Security Protocols* (1999), pp. 172–194.

[20] VIG, J., AND BALLATO, A. Frequency Control Devices. Reprinted from Ultrasonic Instruments and Devices, Academic Press, 1999.

[21] WANT, R., AND PERING, T. New Horizons for Mobile Computing. In *First IEEE International Conference on Pervasive Computing and Communication (PerCom'03)* (Dallas, Texas), pp. 3–8.

Notes

[1] Since infrared channels require line-of-sight links, they cannot be efficiently used for the actual communication between the sensors.

[2] We assume that packets do not carry information that can help identify the source address. Thus we concentrate our efforts on temporal and spatial indistinguishability problems.

[3] Available at: http://www.hpl.hp.com/personal/Jean_Tourrilhes

[4] Note that the spatial indistinguishability property requires that the two devices set the same transmission power. We observed that almost all vendors set it to 15 dBm. Otherwise, the devices should modify their transmission power to a specified value and keep it constant during key agreement.

Reincarnating PCs with Portable SoulPads

Ramón Cáceres Casey Carter Chandra Narayanaswami Mandayam Raghunath

IBM T.J. Watson Research Center
{caceres, chandras, mtr}@us.ibm.com, casey@carter.net
Authors listed in alphabetical order

Abstract

The ability to walk up to any computer, personalize it, and use it as one's own has long been a goal of mobile computing research. We present SoulPad, *a new approach based on carrying an auto-configuring operating system along with a suspended virtual machine on a small portable device. With this approach, the computer boots from the device and resumes the virtual machine, thus giving the user access to his personal environment, including previously running computations.* SoulPad *has minimal infrastructure requirements and is therefore applicable to a wide range of conditions, particularly in developing countries. We report our experience implementing* SoulPad *and using it on a variety of hardware configurations. We address challenges common to systems similar to* SoulPad, *and show that the* SoulPad *model has significant potential as a mobility solution.*

1 Introduction

Today's laptop computers give users two highly desirable features. One is the ability to suspend a computing session (e.g., running applications, open windows) and resume it later, perhaps at a different location. The other is access to their personal and familiar software environment (e.g., applications, files, preferences) wherever they are. In spite of this convenience, a major drawback of this model is that the user has to carry a fairly bulky device. In addition, though docking stations allow the user to use a larger display and attach some peripherals, the user is limited to the capabilities of the hardware integrated in the portable computer, such as the processor and memory.

Before the advent of portable computers, there were two main approaches to suspending a session in one location and resuming it at another. One method was based on process migration between the machines at the two locations [3, 17]. Another technique was to move just the user interface and graphical windows across stationary machines while continuing to run the application processes on a single machine [11, 15]. There are several solutions that store the user's data on a central server to make it possible for a user to log in to one of several machines that are connected to the server and have a common startup environment [16].

More recent solutions to this problem have centered on the use of virtual machines. For example, in Internet Suspend/Resume (ISR) [7, 8] the user's computation state is stored as a check-pointed virtual machine image in the network when computation is suspended, and retrieved from the network when computation is resumed at a machine that has similar base software. ISR has since explored using a portable storage device as a cache [18].

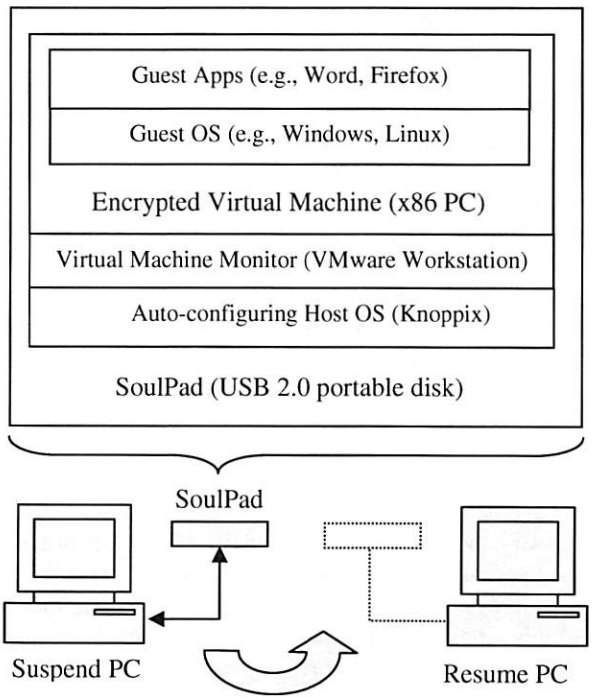


Figure 1: SoulPad architecture and use.

In this paper we present *SoulPad,* a portable device carrying the software stack shown in Figure 1, that allows a user to walk up to a hitherto unseen personal computer and resume a personal computing session that was suspended on another machine. The *SoulPad* approach exploits portable storage devices, fast local wired connections, auto-configuring operating systems and virtual machine technology, while coexisting with the widely deployed PC ecosystem.

In summary, we decouple the user's machine into a body (display, CPU, RAM, I/O) and a soul (session state, software, data, preferences). The soul is carried in

a small and light portable device, the *SoulPad*. The soul can reincarnate on any one of a large class of x86-based personal computers with no preloaded software, and effectively convert that computer into the user's computer. The computers on which the *SoulPad* can reincarnate itself on are denoted as *EnviroPCs*. We presently rely on USB 2.0 connections between the *SoulPad* and the *EnviroPC*. The *EnviroPC's* CPU, memory and I/O devices are used to run the software on the *SoulPad*.

There are several practical advantages to our method. The first is that the *SoulPad* has no battery and thus the user need not worry about recharging it. The second is that no network connectivity is required to retrieve suspended state. Another advantage is that the *EnviroPCs* do not require any preloaded software and thus can be *unmanaged*. In fact, the *EnviroPCs* can be diskless and can be relegated to pieces of furniture that don't require constant monitoring for viruses. Since all software running on the *EnviroPC* comes from the *SoulPad* and belongs to the user, the user does not have to trust a preinstalled operating system on the *EnviroPC*.

Our approach also allows the user to exploit the full capabilities of the *EnviroPC*, for example a high-resolution display or a fast processor. Finally, by resorting to a fast wired connection between the *SoulPad* and the *EnviroPC*, we avoid the problems associated with wireless connections between the two devices – namely device disambiguation and association, and the power consumed by wireless communication as in the Intel Personal Server [21]. We observe that these advantages are in addition to the general benefits of virtualization, such as encapsulation and easier system migration.

We believe that the *SoulPad* approach could change the way computers are built and used. If the software on the internal disks adopt the *SoulPad* stack, users will be able to easily migrate from one machine to another by simply moving the disk. For example, a business professional could insert his disk into a light and compact laptop for travel, into a larger but more powerful laptop for regular use, and even into a wearable computer with an eyeglass-mounted display if necessary. Obviously, the disk attachment interface must be compatible with the different form factors.

Our method is also particularly well suited to developing countries, where a large class of society cannot afford to buy computers and keep them connected to the Internet. Voltage fluctuations and power outages also add to the problem. Shared community PCs provide a solution in such environments. For example many people use web-based applications from public places. However, this solution does not address the personalization and environment preservation issue. By moving to a model where users own the *SoulPad* and borrow or rent the *EnviroPC*, we can reduce their investment and offer them personalization and environment preservation across suspend and resume cycles.

Our approach has only recently become feasible. Technical advances in storage devices have made it possible to carry small disk drives that fit in a pocket and hold upwards of 60GB for around US$150 (in May 2005). Flash storage of several gigabytes already fits on a key fob. Clearly we can expect tens of gigabytes to fit on smaller and cheaper portable and wearable devices over time. Atomic Force Microscope (AFM)-based storage technologies such as Millipede [20] can have a density of 125 GB per square inch ten times higher than the densest magnetic storage available today. Already several portable music players such as the Apple iPod and some digital image viewers feature large drives. Interfaces like USB 2.0 provide sustained data access rates of more than 150Mbps, leading to acceptable resume and suspend times.

Compared with today's laptop model, the disadvantages of our method include the performance degradation due to virtualization, and longer resume times. In addition, portable devices are susceptible to loss or damage, but regularly backing up the contents of the *SoulPad* can address this issue.

We have implemented our solution and report our results in this paper. We address suspend and resume times and how they vary with disk, processor, and interconnect speeds; runtime overheads caused by virtualization and use of an external disk; practical issues that arise due to evolution in processor architecture; and security issues.

We first present the software architecture of *SoulPad*, followed by our implementation and experimental results. We then discuss some of the issues we had to deal with as we moved from concept to prototype, and some challenges that remain. A number of these issues are relevant to other efforts such as ISR [7, 8] and the Stanford Collective [13] that also use virtual machine technology for mobility. Finally, we discuss some of the related work that has helped shape our solution. Throughout this paper, we use the term *SoulPad* to refer both to the design of our system and to any device that embodies that design.

2 Architecture

2.1 Components

We want *SoulPads* to work with a wide range of x86 *EnviroPCs* without relying on a preinstalled operating system. We also want to allow the user to preserve session state across *EnviroPCs*. In order to meet these needs, the software stack on the *SoulPad* has the following three components:

1. A Host OS that boots on *EnviroPCs* and addresses hardware diversity via auto-configuration.
2. A Virtual Machine Monitor (VMM) that can suspend/resume virtual machines and supports Guest OS diversity.
3. A Virtual Machine (VM) that runs the user's applications on a Guest OS of the user's choosing.

While booting on an *EnviroPC*, the auto-configuring Host OS discovers the hardware characteristics and I/O devices of the *EnviroPC*, and configures itself to the hardware present by installing appropriate driver modules. Auto-configuration is a requirement for this layer since the *SoulPad* has to boot on an *EnviroPC* that it may not have seen before. This characteristic contrasts with a traditional operating system that goes through a separate install phase.

Once this step is complete, the Host OS provides a known environment for the next layer, namely the Virtual Machine Monitor. The VMM runs a virtual machine, relying on the underlying Host OS for any services that the VM requires. The VM provides an environment on which the user's operating system and applications (also stored on the *SoulPad*) are run. Since the user's computing environment runs on top of a VM, it is possible for the VMM to suspend the user's session state and resume it later. The suspended session state is also stored on the *SoulPad.*

The user can suspend his session, then shut down the VMM layer and the Host OS, and walk away with his *SoulPad.* The user can later attach the *SoulPad* to a different *EnviroPC*, start the Host OS and the VMM layer, then load the suspended session state, resume it, and continue his session. If the user's tasks do not require network access, the PC may be completely disconnected from the network.

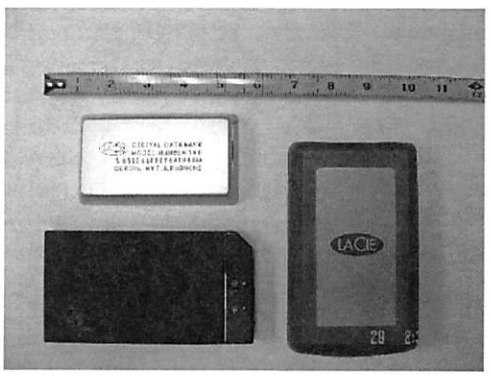


Figure 2: A sample of USB 2.0 portable disks used as SoulPads. Clockwise from upper left: LaCie 40GB DataBank, LaCie 60GB PocketDrive, and IBM 40GB Portable Hard Drive.

The generality of our three-level architecture allows users a choice of personal computing environments, from Windows to Linux to any other OS that can run on the VMs provided by the VMM. Users can even maintain multiple Guest OS environments on the same *SoulPad*, each OS running in its own VM.

2.2 Issues addressed

The issues we addressed in the course of building our *SoulPad* prototype are listed below.

- *Performance:* Working on a VM introduces some overhead when compared to working on bare hardware. Using an external disk instead of an internal disk could make the situation worse. We therefore evaluate the suspend/resume and operational performance of *SoulPad.*
- *Security and privacy:* Portable devices are prone to theft and loss. We safeguard privacy by encrypting the user data stored on a *SoulPad*. Moreover, since the software on *EnviroPCs* may not be trustworthy, we rely only on their hardware and firmware.
- *Reliability:* Portable devices are prone to damage and loss. We implemented a way to recover the contents of *SoulPads* from a backup source.
- *Hardware independence:* There are many hardware differences between PCs. Some of these differences are hidden by VM technology, but others are exposed to the Guest OS and its applications. We need to determine across how wide a range of PCs *SoulPad* will operate.

The following section describes how we addressed some of these issues with our implementation. Later sections will return to discuss these and other challenges in more detail.

3 Implementation

3.1 Overview

We used off-the-shelf USB 2.0 portable disks as *SoulPad* devices. Figure 2 shows some examples. These devices are much smaller and lighter than portable PCs. For example, the LaCie 40GB DataBank measures 4.4 x 2.5 x 0.6 inches and weighs 4.8 ounces. In contrast, a latest-generation "ultraportable" notebook computer like the IBM ThinkPad X40 measures 10.5 x 8.3 x 1.06 inches and weighs 2.7 pounds.

Despite their small size, these portable disks have comparable storage capacity to notebook and laptop PCs, e.g., 40-60 GB. They in fact use the same hard-disk technology. Given the popularity of portable PCs, it follows that *SoulPads* can satisfy the storage needs of large numbers of users.

To implement the software architecture described in the previous section, we made the following choices:

1. Knoppix for the auto-configuring Host OS.
2. VMware Workstation for the VMM.
3. Windows or Linux for the Guest OS.

Knoppix [24] provides us with the zero-install and auto-configuration features that *SoulPad* needs from a Host OS. Knoppix is a version of GNU/Linux distributed as a single bootable CD that includes the Linux kernel and a range of applications. Knoppix enables users to get a familiar Linux desktop along with their favorite applications on almost any PC without having to install any software on the PC's hard disk.

The bootloader from the CD loads a Linux kernel and an in-memory disk image called the *Initial RAM Disk*. Subsequently, Knoppix scans for devices, loads the appropriate device drivers, initializes discovered network interfaces, generates an appropriate X11 configuration for the discovered display hardware, and carries out other auto-configuration steps. These steps are necessary because Knoppix does not have prior knowledge of the hardware configuration of PCs on which it boots.

An in-memory filesystem is created for read-write data. All of the applications, libraries and other read-only data reside on a compressed filesystem on the CD, which is mounted using a loopback device in the kernel. The compressed filesystem approach enables Knoppix to pack almost 2 Gigabytes of data onto a single 700MB CD. All of the local session state created by the user typically resides in the in-memory filesystem and is lost when the user shuts down Knoppix. Some users combine a Knoppix CD with a small USB flash key where they store their personal files and other persistent data.

We create a *SoulPad* disk by first installing Knoppix on a USB hard disk, using the hard-disk install script that comes with Knoppix. We also install a bootloader on the USB disk that loads the kernel and the Initial RAM Disk in the same manner as the bootloader on a Knoppix CD. We had to make a few modifications to the Initial RAM Disk and startup scripts, for example to ensure that USB-related kernel modules were loaded before trying to mount the root file system from the USB disk. With these changes, we were able to take the USB disk from one machine to another and boot a Knoppix environment.

While Knoppix by itself enables users to walk up to any PC and personalize it with their Linux environment, there is no easy way for users to preserve their computing state as they move from one machine to another because Knoppix needs a full reboot every time it is moved. Knoppix users are also limited to that one OS.

We installed VMware Workstation [24] on top of Knoppix to support suspend/resume of user sessions as well as OS diversity. We then created virtual machines on which we installed Windows XP Professional or a Linux variant as the Guest OS.

We automated the *SoulPad* suspend and resume sequences so that each runs to completion after an initial user action. Users initiate suspend by selecting the VMware Workstation suspend operation on their screens. After the VM suspends, Knoppix shuts down and powers down the machine. At this point the user can disconnect the *SoulPad* from one PC and connect it to another. Users initiate a resume operation by powering up the new PC so that it boots from the *SoulPad*. The PC boots into Knoppix, which starts VMware Workstation, which resumes the Guest OS session.

On our *SoulPad* disks we created a 4GB partition to hold Knoppix and a 2G partition to serve as swap space. The remaining disk space is available for sharing among VM images. For example, on a 40GB disk holding only one VM, 34GB are available for the Guest OS environment.

3.2 Encrypted virtual machine image

To protect user data if a *SoulPad* is misplaced or stolen, we encrypt the disk partition that holds the VM images using the AES128 block cipher. We used the publicly available `loop-aes` package for Linux in our implementation.

The encryption key is generated by hashing a user-supplied passphrase. After the Host OS boots, it prompts the user to enter the passphrase. If the user supplies an incorrect passphrase, the resulting hash will not correspond to the AES key and the mount operation will fail since the decrypted data will not correspond to a valid filesystem. In order to defeat brute force attacks that attempt to guess the passphrase, the `loop-aes` package requires the passphrase to be at least 20 characters long. For convenience, we permit users to supply this passphrase via an auxiliary USB flash key. While the Guest OS partition is mounted, the AES key is retained in kernel memory. When the partition is unmounted, the AES key is erased from memory. It is never stored on disk.

At run time it is possible that the Host OS swaps out pages holding the user's Guest OS state to the swap partition on the *SoulPad*. We also use `loop-aes` to encrypt the swap partition to prevent user data from appearing in plaintext form on the *SoulPad*. The key for the swap partition is auto-generated for each session since swap state does not have to be preserved across Host OS boot cycles.

SoulPad never writes to the internal disk on an *EnviroPC*. Therefore, there is no risk of leaving

sensitive data on the PC's persistent storage after disconnecting.

3.3 Networking configuration

At resume time, if the *EnviroPC* is connected to a network with a working DHCP server, the Host OS will obtain an IP address and establish network connectivity. The VMware Workstation virtual machines are configured to use Network Address Translation (NAT) to connect to the external network through the Host OS. Thus, the Guest OS enjoys network connectivity whenever the Host OS does. In short, from a networking perspective, a *SoulPad* suspend followed by a resume is similar to suspending a laptop at one location and resuming it at another location.

Many networked applications already support suspend and resume of laptops, e.g., email and instant messaging clients. They simply attempt to re-establish their network connections at resume time. Similarly in the case of *SoulPad*, such applications running inside the Guest OS re-establish their connections when they are able to do so. In some cases, the resume may happen outside an intranet and some network resources may not be reachable unless the user establishes a VPN connection into the intranet. Laptop users are already familiar with such situations and the behavior is identical while using a *SoulPad*.

3.4 Backups

In our enterprise environment we have configured backups from the *SoulPad* to Tivoli Storage Manager (TSM), a file-level networked backup service. Whenever the *SoulPad* is connected to a PC that has connectivity to the TSM server, we perform an incremental backup of the *SoulPad*. If a user loses his *SoulPad*, a copy of it can be re-created from the backup server. Again, this model is similar to the situation where a user loses his laptop and has to recover data from the most recent backup.

On our prototype *SoulPads,* we have configured incremental backups both at the Host OS and Guest OS levels. At the Host OS level any changes to Guest OS files appear as changes to the large binary files corresponding to the VMware Workstation virtual disks. Our current backup implementation does not handle minor modifications to large binary files very well, as it simply treats the file as having changed and transfers the entire file to the backup server. So, we specifically exclude these files from the files backed up at the Host OS level. Instead we rely on the incremental backups at the Guest OS level to back up the modified Guest OS files. In the future we propose to investigate better backup schemes that handle large binary files.

We do not backup the suspended virtual machine state at suspend time because this would add considerable latency to the suspend operation. This means that if the user loses the *SoulPad,* he also loses the latest session state and must reboot the VM after recovering *SoulPad* state from backup.

In environments with poor infrastructure where managed network backup services are not viable, it is possible to perform local backups using LAN-connected devices such as Mirra [24], or backups to a second locally connected USB storage device.

4 Experimental Results

We confirmed the usability of *SoulPad* through a variety of experiments. These experiments fall into three main categories: resume and suspend latencies, application response times, and hardware independence. This section describes our methodology and results.

The *SoulPad* software stack used in all experiments consisted of Knoppix 3.4, VMware Workstation 4.5.1, and Windows XP Professional. In addition, VMware Tools was installed in the Guest OS of all VMs.

Disk Model	Type	Size (GB)	Speed (RPM)	Transfer Rate (MB/sec)
(internal)	IDE	40	7200	**44.92**
PocketDrive	USB	60	7200	**23.09**
DataBank	USB	40	4200	**18.45**
MobileDrive	USB	40	4200	**8.13**

Table 1: Characteristics of disks used.

We used disks with varying characteristics, as shown in Table 1. The transfer rates are averages of 10 runs of the `hdparm -t` Linux command. This command measures how fast the drive can sustain sequential data reads, without file-system buffering effects. All transfer rates were measured on a NetVista desktop PC, using a USB 2.0 connection for all but the IDE disk.

As shown, the IDE disk has close to twice the transfer rate of the fastest USB disk. There are also large differences in transfer rates among USB disks.

4.1 Resume and suspend latencies

Resume and suspend latencies are key to *SoulPad*'s usability. We define resume latency as the time between when the user powers up the *SoulPad-EnviroPC* combination, and when the VM has finished resuming, i.e., when the user can continue working. We define suspend latency as the time between when the user requests that the VM be suspended, and when the Host OS has saved modified state to the *SoulPad* and shut down, i.e., when the user can walk away with his *SoulPad.*

Disk Model	PC Model	CPU (GHz, Pentium Family)	Physical Memory Size (MB)	Inter-connect Type	Resume Time Average (sec)	Resume Time Std Dev (sec)	Suspend Time Average (sec)	Suspend Time Std Dev (sec)
(internal)	NetVista	2.4, IV	1024	IDE	**116**	1.0	**10**	0.4
PocketDrive	NetVista	2.4, IV	1024	USB 2.0	**121**	4.3	**26**	1.2
DataBank	ThinkPad T41	1.7, M	1024	USB 2.0	**134**	0.6	**26**	0.3
DataBank	NetVista	2.4, IV	1024	USB 2.0	**141**	1.5	**22**	0.4
MobileDrive	NetVista	2.4, IV	1024	USB 2.0	**170**	2.6	**30**	0.2
MobileDrive	ThinkCentre	3.0, IV	512	USB 2.0	**179**	7.4	**50**	2.0
MobileDrive	ThinkPad T30	2.4, M	1024	USB 1.1	**977**	34.5	**372**	29.2

Table 2: Resume and suspend latencies, sorted by increasing resume time.

We designed our suspend/resume experiments to expose the effects of disk speed, interconnect speed, processor speed, and memory size. Table 2 shows averages and standard deviations calculated over at least 10 runs for a variety of disk and PC configurations. The NetVista and ThinkCentre PCs are desktop machines; the two ThinkPad models are laptops. In all these experiments, there were 256MB of memory and 16GB of disk space allocated to the virtual machine. In the interests of simplicity and space, we omit results for hardware combinations that do not expose significant additional information.

The first row of Table 2 serves as a reference point for additional observations. For the results in this row, we installed the *SoulPad* software stack on the internal disk of the NetVista instead of on a portable disk.

It is noteworthy that external USB drives achieved resume times close to those of the internal IDE drive. For example, the average resume time on the PocketDrive connected to the same NetVista PC was only 5 seconds longer than the reference (121 vs. 116 seconds). The average suspend time on that same configuration was 16 seconds longer than the reference (26 vs. 10 seconds). Disks with lower transfer rates and rotational speeds, like the DataBank and MobileDrive, have longer resume and suspend times. Other disk characteristics not captured in Table 1, such as buffer size, also affect the resume and suspend latencies shown in Table 2.

Another observation is that physical memory size matters for *SoulPad*. Resume and suspend latencies are noticeably longer on the ThinkCentre PC with 512 MB less memory than the other PCs, even though the ThinkCentre has the fastest CPU. Resume time rose to nearly 3 minutes and suspend time closer to 1 minute.

Finally, the last row of Table 2 makes clear that USB 1.1 is too slow to support *SoulPad*. Resume times when using USB 1.1 rise to more than 16 minutes while suspend times rise to more than 6 minutes.

Our overall conclusion is that *SoulPad* is usable on a range of existing portable disk and PC configurations. Disk transfer rate and physical memory size have a significant effect on resume and suspend latencies, while processor speed has less of an effect (at least for the 1.7-3.0 GHz range we used in our experiments). Disk-to-PC interconnects with speeds comparable to USB 2.0 are required but increasingly standard on commercially available PCs.

We proceeded to collect fine-grained timings of different stages in the *SoulPad* suspend and resume sequences. As a timing mechanism, we used the Time Stamp Counter (TSC) available on x86 processors. This monotonically increasing value resets to zero on each powerup, advances with each clock cycle, and can be read by a single instruction from firmware, the boot loader, kernel space, or user space.

Table 3 and Table 4 contain timings captured during sample resume and suspend runs, respectively. Both runs used the DataBank disk connected over USB2.0 to the NetVista PC, as in the fourth row of Table 2.

As shown in Table 3, the sample *SoulPad* resume operation took almost 140 seconds. Autoconfiguring the Host OS accounted for somewhat less than half of this time, or roughly 57 seconds. We will see below that there is room for reducing the latency of this stage.

Resume Stage	Individual Time (sec)	Cumulative Time (sec)
BIOS Power-On Self-Test	16.13	16.13
Boot Loader	1.02	17.15
Host OS Kernel Startup	5.79	22.94
Host OS Init RAM Disk	2.59	25.53
Host OS Autoconfig	56.83	82.36
VM State Load + Resume	57.27	139.63

Table 3: Resume stages and sample latencies.

Suspend Stage	Individual Time (sec)	Cumulative Time (sec)
VM Suspend	5.80	5.80
VM State Save	15.79	21.59

Table 4: Suspend stages and sample latencies.

The time to load VM state from disk into memory and resume the running VM is another major contributor to total resume latency, also accounting for roughly 57 seconds in the sample run of Table 3. Techniques such as *ballooning* [13] can be used to reduce the latency of this stage. Ballooning zeroes unused pages of physical memory allocated to VMs. These pages would then lend themselves to more effective compression for transfer between *SoulPads* and PCs.

As shown in Table 4, the sample *SoulPad* suspend operation took under 22 seconds. The two main components of this latency are the time to stop the VM (roughly 6 seconds) and the time to save to disk the contents of the VM's memory as well as other recently changed VM state (roughly 16 seconds). Aside from the contents of the VM memory, the amount of state saved at suspend time is relatively small because writes to the VM's virtual disks have been propagated to the *SoulPad* throughout the VM's operation.

We then explored ways to reduce the resume latency by streamlining the Knoppix autoconfiguration procedure. The results in Table 2 were obtained using a base Knoppix installation. We were able to eliminate two steps from this base case: rebuilding the mapping from library names to path names, and rebuilding kernel-module dependencies. The former is necessary only when libraries are installed or moved, and the latter is only necessary when kernel modules are added, changed, or removed. Such Host OS configuration changes will be rare on a *SoulPad* since the Host OS is only used as a vehicle to bring up a virtual machine. This layer of the *SoulPad* architecture can be tightly managed by system administrators working for enterprises or service providers.

Experiment	Resume Time Average (sec)	Resume Time Std Dev (sec)	Suspend Time Average (sec)	Suspend Time Std Dev (sec)
Original	**141**	1.5	**22**	0.4
Streamlined	**129**	1.6	**22**	0.3
Encrypted	**139**	1.3	**28**	0.6

Table 5: Impact on resume and suspend latencies of streamlining the Host OS boot sequence, then storing the VM image in an encrypted file system.

Table 5 shows the impact on resume latency of eliminating these two steps. These measurements were done on the same DataBank-NetVista combination shown in the fourth row of Table 2, yielding a baseline resume time of 141 seconds. As shown in Table 5, streamlining the Knoppix autoconfiguration stage reduced resume latency by 12 seconds, to 129 seconds total. Further optimizations of the boot sequence may be possible.

Table 5 also shows the impact of encrypting the VM image. We placed the VM image on a file system encrypted with the AES128 cipher. We then measured suspend and resume latencies on the same DataBank-NetVista combination after streamlining the Knoppix autoconfiguration stage. As shown, resume latency rose by 10 seconds but remained below the original 141 seconds, and suspend latency rose by 6 seconds but remained below 30 seconds. We conclude that using an encrypted file system is both desirable and viable.

Finally, it is useful to compare *SoulPad* suspend/resume times to hibernate/resume and shutdown/boot times on today's portable computers. Hibernation saves the session state to disk before powering down the machine. Resume after hibernation restores the session from disk. Shutdown and boot are familiar operations common to all PCs. Portable PCs also offer a standby/resume feature, but it cannot hold session state for arbitrary periods of time. Standby holds state in volatile memory and draws battery power.

We measured a ThinkPad T41 running Windows XP from its internal IDE disk. Over three runs, hibernate and resume times both varied between 26 and 28 seconds, shutdown times varied between 31 and 43 seconds, and boot times were stable around 50 seconds.

In comparison, on that same ThinkPad T41 running off the DataBank disk, Table 2 shows that *SoulPad* suspend times averaged 26 seconds and resume times 134 seconds. We see that *SoulPad* suspend times are roughly equal to hibernate and shutdown times, although the user must always wait for the *SoulPad* suspend sequence to complete before walking away. *SoulPad* resume times are considerably longer than resume-after-hibernate and boot times. The extra waits are the price to pay for the added portability and hardware independence of *SoulPad*. However, these suspend/resume latencies can be expected to improve with continuing hardware and software advances.

4.2 Application response times

Application response times are another key metric of *SoulPad*'s usability. The time it takes for applications to respond to user-initiated operations is a measure of what it feels like to use the system for everyday work.

We used SYSmark 2002 [22], an industry-standard benchmark suite, to evaluate the overhead introduced by the *SoulPad* three-level architecture when compared to a standard OS installation running on bare hardware. SYSmark employs workloads that emulate common uses of Windows PCs in business environments. The workloads fall into two classes: Office Productivity and Internet Content Creation. Office Productivity exercises nine applications that include programs in the Microsoft Office suite and McAfee VirusScan. Internet Content

Creation exercises five applications: Adobe Photoshop, Adobe Premiere, Macromedia Dreamweaver, Macromedia Flash, and Microsoft Media Encoder.

SYSmark measures the time it takes for applications to complete tasks initiated by mouse clicks or keystrokes. It uses Visual Test and Visual Basic to emulate a person sending commands to the computer. The sequence of commands was chosen by observing industry professionals at work. Additional detail on the workload generation and performance measurement methodology is available from the SYSmark 2002 documentation [22].

At the end of a run SYSmark reports the average response time over hundreds of operations. Given the variety of these operations, from replacing a word in a text document to re-encoding a video clip, it is more meaningful to examine the relative response times between system configurations than the absolute values. In our results we thus normalize response times to a baseline system configuration. All our SYSmark runs used the NetVista PC with 1 GB of memory installed in the physical machine, and 512 MB of memory allocated to the VM when a VM was used.

System Configuration	Office Productivity Workload	Internet Content Creation Workload
Physical, IDE	**1.00** (0.004)	**1.00** (0.011)
Virtual, IDE	**1.29** (0.019)	**1.26** (0.005)
Virtual, USB	**1.42** (0.025)	**1.38** (0.023)

Table 6: Normalized response times (with standard deviations) as reported by SYSmark 2002.

Table 6 show response time averages (with standard deviations in parentheses) across three runs of the Office Productivity and Internet Content Creation workloads. The row labeled "Physical, IDE" represents our baseline configuration: Windows XP Professional running on the physical machine and from the internal IDE drive. The row labeled "Virtual, IDE" corresponds to running the *SoulPad* three-level architecture, but still from the internal IDE drive. It is intended to isolate the overhead of virtualization from the overhead of using an external drive. The row labeled "Virtual, USB" corresponds to running the *SoulPad* architecture on the PocketDrive connected via USB 2.0. This last row also includes the overhead of storing the VM image on an encrypted file system.

Our overall conclusion is that *SoulPad* is usable on today's portable disks and PCs, and it will become increasingly usable as hardware continues to improve. Across the two workloads, moving to a VM-based configuration on the IDE drive incurred a 26-29% increase in response time. Moving to a VM-based configuration on the USB drive incurred a 38-42% increase over the baseline. These overhead numbers seem high, but with today's fast disks and PCs, absolute performance remains acceptable for a large class of business and personal users. It is interesting to note that this slowdown is roughly equivalent to using a one year-old machine without virtualization. As further anecdotal evidence, one of the authors has used VMware Workstation on laptop-class PCs since 2000, and found its performance acceptable for everyday work. User-perceived performance can also be expected to improve along with hardware and software advances.

4.3 Hardware independence

A remaining usability question is: across how wide a range of PCs will *SoulPad* work? We ran experiments on a larger collection of PCs than mentioned so far in order to explore these hardware independence issues. Among these additional systems were six models of IBM ThinkPad laptops, four models of IBM ThinkCentre desktops, and a Dell Dimension desktop. While *SoulPad* worked smoothly on the earlier set of machines, we ran into a number of practical obstacles when we expanded the set.

One problem is that not all PCs would boot *SoulPad* from USB. Increasingly, new PCs include a BIOS option to boot from USB mass storage devices. However, many legacy PCs do not offer this option. Furthermore, the option is not uniformly easy to use on PCs that do offer it. Sometimes booting from USB must first be enabled. Once enabled, the USB option must be placed in priority order with respect to other boot devices. On some PCs, it is necessary to reposition the USB option in the priority order every time a different USB device is attached, or even on every boot operation. Only on some recently manufactured PCs is it possible to position the USB option once for all time and all devices, so that the PC will always boot from a *SoulPad* when one is connected.

We worked around these difficulties by creating an auxiliary mini-CD containing a boot loader that quickly switches the boot sequence to a USB device. Booting from CD is universally supported on current-generation PCs, usually higher in priority than booting from internal disk. Using this CD we were thus able to use *SoulPad* on all the PCs we tried. *SoulPad* users could carry such an auxiliary CD for use when booting from USB is not available or properly configured. However, over time we expect that booting from USB will attain the ubiquity and ease of use of booting from CD.

Another problem is that *SoulPad* was not always able to resume a user session with the same graphics configuration in use at suspend time. Graphics settings are personal choices determined by factors like visual acuity and application mix. Graphics capabilities vary

widely among PCs. For example, some machines support display resolutions as high as 1600x1200 pixels, while others only 1024x768.

As a result, it was sometimes necessary to manually change graphics settings after a resume operation. For example, the display resolution and color depth settings in the Guest OS must match those of the Host OS in order for a VM to use the full screen, and not be limited to running inside the Host OS window allocated to the VMware Workstation application. It is possible to work within that window, but it is often preferable to cede the whole display to the Guest OS, for example to hide from naïve users that they are dealing with a virtual machine. Further work is needed to automatically adapt the overall graphics configuration for best effect.

A final problem we ran into occurred when a PC did not have enough memory to run the VM stored in a *SoulPad*. Our default *SoulPad* configuration allocates 256MB to the VM. These VMs would not resume on PCs with 256MB or less of total physical memory. In principle the VMM should be able to swap portions of the VM as necessary to operate with smaller memory sizes (at the expense of performance), but current VMM implementations have limitations in this area. Ideally, the VMM and Guest OS should adapt to the amount of physical memory available at resume time.

The above experiences make clear that work remains before the full promise of VM-based mobility, where "any PC is your PC", can be realized. However, it is important to note that such mobility is already practical if constrained to a known subset of PC models and configurations, as is the case in many enterprise settings. In addition, with some planning and testing, any single PC manufacturer can ensure that a *SoulPad*-like solution works on all its offerings. The following section discusses additional hardware independence issues.

5 Discussion and Future Work

5.1 Instruction Set Architecture (ISA) diversity

The x86 architecture has steadily evolved over the years with the addition of instructions to meet the demands of applications such as media processing, security, etc. Table 7 shows the changes to the Intel ISA between 1997 and 2004. For performance reasons, today's VM technologies permit guest operating systems to query the hardware and determine which instructions are supported, and directly execute the instructions that are available. When we use VM technology for mobility we encounter the situation of suspending a VM on one machine and resuming it on another machine that may not support the same set of instructions. For instance a VM may be suspended on a Pentium IV and resumed on a Pentium III. The Guest OS and applications may have queried the hardware on the Pentium IV, determined that SSE3 instructions are available, and subsequently try to execute them on the Pentium III where such instructions are not supported. Since the *SoulPad* moves the Host OS and the VMM layers from machine to machine, these layers have to deal with different ISAs as well.

Processor	Year	Feature
Pentium	1993	Baseline.
Pentium II MMX	1997	**MMX:** SIMD integer operations. No new registers. 57 new instructions.
Pentium III, Celeron	1999	**SSE:** Streaming SIMD extensions. 32-bit parallel floating point arithmetic support. 8 new 128-bit registers. 70 new instructions.
Pentium IV, Celeron II	2002	**SSE2:** 64-bit parallel floating point arithmetic support. 144 new instructions.
Pentium IV Prescott, Pentium IV M, Celeron D	2004	**SSE3:** Complex arithmetic. Memory and thread handling. 13 new instructions.

Table 7: Intel processor evolution.

5.1.1 Solutions

The approach to handling ISA diversity varies by layer. Let us first consider the Host OS. Since ISAs are typically backward compatible, the Host OS (kernel and other executables) can be configured for an older processor family such as 386 and be expected to work on newer processor families.

This approach does not use the newer instructions and could result in lower performance. This drawback can be addressed by keeping multiple copies of ISA-dependent components of the Host OS, each configured to a different ISA, and choosing the appropriate version at boot time. Since the only function of the Host OS is to run the VMM we do not believe the storage overhead for carrying multiple versions of these components will be significant.

Similar approaches can be applied for the VMM as well. When the VMM is started the version corresponding to the ISA of the *EnviroPC* can be used to ensure that the VMM itself does not use any unsupported instructions.

The matter is much more complicated for the VM image, because we need to suspend and resume the Guest OS and applications without restarting or reinstalling. If operating systems and applications could dynamically adapt to changes in the ISA, the VMM could simply signal the Guest OS that the ISA has

changed and expect the adaptation to take care of this problem.

While such dynamic adaptation to ISA changes is a hard technical challenge, there have been several examples where applications have evolved the ability to survive changes in the environment. For instance, in the early stages of graphical windowing systems, applications were written to fixed display resolution and window sizes. However, today most applications dynamically adapt to changing graphics configurations. Similarly, many applications have been made resilient to changes in network settings. Several server operating systems have developed the ability to dynamically reconfigure themselves (without rebooting) to changes in hardware resources such as amount of DRAM, number of CPUs, and I/O devices.

One way of handling dynamic ISA changes is to compile applications such that the performance critical sections of the code are built for different ISAs, while the other sections are built for the oldest ISA. At run time, applications choose the ISA-specific code paths in the performance-critical sections. When a suspend request is received, applications complete the current performance-critical section before allowing suspension. At resume time, the application probes the ISA so that subsequent performance-critical sections take the paths corresponding to the new ISA. Such an approach can also be used in the Host OS and the VMM layers to avoid carrying multiple versions.

In summary, all layers of software need to be aware that they may be suspended on one processor family and resumed on another. They should adapt to the change if performance benefits of newer processor architectures are to be exploited and mobility across a larger class of machines is desired.

5.1.2 Practical workarounds

Since dynamic adaptation to ISA changes is not currently supported by PC operating systems, compromises are necessary to use VM technology for mobility.

At present there is a tight binding between the software and the ISA it runs on. The binding typically happens at compile time or install time, though it can happen later as well. If the binding happens only at compile or install time we could install the entire guest VM software on an old processor family. If all the *EnviroPCs* encountered by the user are the same or newer, none of the unsupported instructions will be encountered. At the same time none of the performance enhancements possible with the newer ISA will be used by the guest VM. If the user were to try to use the *SoulPad* with an ISA that is older than the one configured, the Host OS should refuse to resume the VM. This workaround would limit the range of the *EnviroPCs* that can be used. While this approach works, it delivers lower performance.

If the Guest OS and applications bind to the ISA at start time instead of install time, the VMM may ask the user to reboot the Guest OS or restart the appropriate applications when a different ISA is encountered. This approach will exploit the highest performance offered by the current ISA, but will result in lower usability.

Another method is to be more aggressive and configure the guest VM software for the most recent processor architecture. When running on an older machine and an unsupported instruction is encountered, the VMM should emulate it using available instructions. To the best of our knowledge, today's VMMs do not emulate missing instructions. If this feature was added, a penalty is paid only when the user visits older machines. If applications bind to the ISA at start time, it may be advantageous to restart them on older ISAs instead of relying on instruction emulation support provided by the VMM. If instruction emulation support is provided in the VMM, it is important to ensure that when newer instructions are added to the ISA, emulation support is made available before the Guest OS or any applications start using those instructions.

5.2 Software licensing

Machine virtualization disrupts traditional approaches to software licensing. For instance, the arguably dominant licensing model for PC software permits use of the software "on a single computer". In the absence of VM technology this model is relatively easy to understand. However, the introduction of VM technology, VM-based mobility, and *SoulPad* raises difficult questions, such as:

1. Should the term "computer" be taken to mean a virtual or physical machine?
2. Should the term "computer" be taken to mean the processor that runs the software or the storage device that holds the software?
3. Should users be allowed to run the software on two or more VMs on the same physical machine?
4. Should users be allowed to run the software on a VM that moves between physical machines?

Not only does the language of licenses need to change to make clear the answers to such questions, but software that enforces licenses must also change accordingly. Some software products incorporate code to verify that the user is in compliance with the license terms. A popular form of this code records information about the hardware present at install time, and periodically checks at run time whether the "computer" has changed significantly since the install operation. As

we've discussed, VMs do not hide all aspects of physical hardware from Guest OSs. Therefore, software installed on a *SoulPad* may stop working when moved between significantly different physical machines.

For VM-based mobility in general, and *SoulPad* in particular, it is important that software vendors recognize the above issues and devise license terms that accommodate user needs. Some software vendors are already aware of the disruptive nature of VM technology and have started addressing these issues.

5.3 Developing countries

SoulPad is well suited to developing countries. One reason is that it can work on disconnected PCs, so that it is a good fit for environments with poor networking and server infrastructure. Another reason is that it supports personalized computing on shared PCs.

Although PC prices continue to drop, in many parts of the world there is more to owning a PC than buying the basic computer. Electric power to homes often suffers from voltage fluctuations and complete outages, forcing the purchase of an uninterruptible power supply to ensure clean software shutdowns and protect the hardware from damage. In addition, many people cannot afford to provide Internet service to their homes. Sharing PCs maintained at a community center or Internet café remains popular because it amortizes these infrastructure costs across multiple users.

SoulPad allows users of such shared PCs to maintain widely different software environments without interfering with each other. Furthermore, *SoulPad* users do not store sensitive data on shared machines. Users maintain physical control over their software and data, while the inherent portability of the solution prevents over-reliance on any particular infrastructure provider.

5.4 Security and privacy

We have lowered the security and privacy risks for *SoulPad* users in several important ways. One, we start *EnviroPCs* from a known power-down state. Two, we do not run any software previously installed on *EnviroPCs*. Three, we encrypt the VM image and swap space on *SoulPads*. Four, we do not write anything to stable storage in *EnviroPCs*. Five, we power down the *EnviroPC* at the end of the suspend procedure to let the memory contents dissipate. Some versions of Knoppix also deliberately wipe memory as part of normal shutdown procedures.

However, some security and privacy concerns remain. For example, we have not protected against compromised firmware (e.g., the BIOS). We have also not addressed hardware-related threats such as key loggers or bus snoopers. A future *SoulPad* with modest processing capacity could query the Trusted Platform Module (TPM) [23] hardware, increasingly available in commodity PCs, to determine that the BIOS is trustworthy before allowing the Host OS to boot.

In addition, equipping *SoulPads* with biometric authentication hardware would be an improvement over our current practice of asking for a passphrase through the *EnviroPC*. Note that it is not necessary to equip *SoulPads* with a battery to support biometric authentication or TPM-based attestation. Power for these operations can be obtained from the *EnviroPC*.

The *SoulPad* model also presents security risks for owners of *EnviroPCs*. Since the Host OS boots directly on the PC, a *SoulPad* user has complete control over the PC, including any internal disks and network interfaces. A malicious *SoulPad* user could damage the contents of a hard disk, or launch a network-based attack.

Fortunately, there are ways to prevent *SoulPads* from modifying internal disks. For example, PC owners could password-protect the disks using facilities already provided by most disks and BIOSes on current PCs. In addition, *EnviroPCs* can be diskless as described earlier. Addressing the potential for network-based attacks remains an area for future work.

6 Related Work

Users invest large amounts of time personalizing their computing environments and setting up computing sessions. They naturally desire to reuse as much of these environments and sessions as possible. Solutions to meet this desire have evolved in stages. At first, users of time-shared machines were satisfied with the ability to reuse data files created during earlier sessions. As applications became more complex and long-lived, researchers also attempted to move running processes across machines for load balancing, single-application mobility, etc. Examples of such efforts include the V System [17], Butler [10], Condor [9], and Sprite [3].

As computers became more plentiful, users felt the need for a familiar look and feel, i.e., common file systems, desktop look and feel, application preferences, etc., across different machines. Distributed file systems such as NFS, AFS, etc., and networked stations such as X terminals and Windows terminals, allowed users access to remote files and applications from different machines. We have now reached the point where sessions last days or months and users want the ability to suspend a complete working session, i.e., the complete desktop, at one machine and resume at another exactly where they left off. Users also want automatic remapping of peripherals in different

settings, such as reassigning the default printer to a local printer in the new location.

The related work towards the goal of suspending and resuming a complete session can be split into two broad categories, one that required the user to carry something with them and another that did not.

We first look at solutions that did not require the user to carry any device with them. We note that all of these approaches require common software to be installed on the machines. Some approaches like Teleporting [12] and XMove [15] intercept the data flowing between the server and the client at a medium-grain graphics protocol level such as the X protocol. The graphics interface is moved to a different machine and the application continues to run on the same remote machine. The X Server is switched to another client by using a proxy X Server in between the GUI and the remote machine. The proxy is needed to handle the differences in capabilities of the two clients, such as color depth of the frame buffer, color-map tables, etc. Sluggish performance due to high network latencies and exposures in the X security model limited this approach.

Another approach drew the line between the application and the graphics I/O at a lower level. Stateless thin clients such as SLIM [16] (elements of which were incorporated into *SunRay* [24]) and VNC [11] included low-level graphics primitives that operated on the frame buffer, such as bitmap transfers. Applications ran on servers and user input was provided at the thin client. A high-bandwidth connection between the thin clients and servers was necessary for good interactive performance. Users could access their applications from different machines and resume where they left off previously, since all state was maintained at the server. The InfoPad [19] used a similar partition between the application and the GUI, but realized it on a small portable device that included wireless connectivity between it and the server.

Chen and Noble [2] observed that virtual machine technology [4] can be used to migrate sessions between computers and thus be used for mobility. Internet Suspend/Resume [7,8] developed this idea and demonstrated that using commercial VM technology such as VMware Workstation, together with a networked file system such as Coda [14], it is possible to walk up to a machine and resume a suspended session. Each ISR client has a Host OS and VMware Workstation preinstalled, and has access to a networked repository of VM images. When a session is suspended the VM image is stored on a server. When a user resumes a session, this image is restored and the session is continued. This model is elegant because the user need not carry a device. However, the host machines need network connectivity and preloaded software.

More recently ISR has added portable storage as a lookaside cache to speed up resume time [18]. The copy in the network is considered the primary copy and the portable copy is the secondary copy. The ISR project has independently experimented with running VMware Workstation over Knoppix from a portable storage device, for the purpose of introducing ISR to users without disturbing the internal disks on their machines [5]. Sapuntzakis et al. [13] have studied how to optimize the transfer of encapsulated virtual machine state between different machines. VMotion by VMware [24] is a commercial product that allows migration of virtual machines across different physical machines.

MobiDesk [1] transparently virtualizes the display, operating system and network. Applications run on hosting servers. The clients are input and output devices. Individual processes are virtualized instead of the complete operating system environment. The display is virtualized with a virtual display driver which is similar in flavor to that in SLIM and VNC. Network connections operate through a proxy and a technique similar to NAT is used to map fixed virtual address to new physical network addresses.

The ability to carry a portable computer, suspend the machine, and resume work at another location has marked the most widespread realization of user mobility. Suspend/resume times were acceptable and people could work disconnected. Network connections had to be re-established and applications needed to be resilient. For example telnet sessions are not preserved. Technologies such as MobiDesk [1] consider how to preserve network connections across session suspend and resume. Efforts to build smaller portable machines with similar functionality include the IBM MetaPad [24], the Antelope Modular Computing Platform [24], the OQO [24], and several wearable computers. The drawback with these approaches is that the user is limited to the capabilities of his portable computer, and cannot leverage more powerful CPU and memory resources even if available. The Intel Personal Server [21] utilizes other devices near it over wireless links and web-based interfaces to view data. The IBM Personal Mobile Hub [6] acts as an intermediary between body worn sensors and back end infrastructure.

Some recent commercial offerings attempt to support personalization of anonymous PCs. For example, Migo [24] allows the user to carry personal settings and files on a USB flash key. One limitation of this approach is that it must be tailored for each application to be migrated. MetroPipe [24] starts a CPU emulator on top of an existing OS, then boots a tiny variant of Knoppix. It provides personalized access to networked services, but does not preserve user sessions.

Name	What is carried?	Host PC dependencies	Key benefits	Some drawbacks
Laptop computer	CPU, disk, memory, screen, kbd, battery	(Not Applicable)	Ubiquitous and proven	Size and weight. Needs separate backup.
Ultraportable e.g., OQO	Same as laptop	(Not Applicable)	Smaller and lighter than laptops.	Small display and keyboard. Needs separate backup.
Portable preferences, e.g., Migo	Personal settings in portable storage	Identical OS and application software installed	Small amount of data to carry	Needs per application analysis. Dependency on Host PC software configuration. Session state is not preserved.
MetaPad, Antelope	CPU, memory, disk, suspend battery	Custom connector to I/O peripherals	Adapts to multiple form factors	Compute capacity limited to resources on portable. Needs separate backup.
SLIM, VNC, XMove	Nothing	Needs preinstalled software	Works on many different platforms	Needs reliable low-latency network
MobiDesk	Nothing	Needs preinstalled software	Both processes and I/O can be moved	Needs reliable low-latency network
ISR w/o portable storage	Nothing	Needs preinstalled software	Local execution hides network latency	Needs high-bandwidth network for fast resume at arbitrary locations
ISR with portable storage	Portable storage device used as a cache	Needs preinstalled software	Fast resume and suspend	Needs network to validate cache
SoulPad	Complete SW environment on portable disk	USB 2.0	No preinstalled software	Increased resume time due to Host OS auto-configuration. Needs separate backup.

Table 8: Comparison of mobility approaches.

Even after restoring personal settings and suspended sessions, it is necessary to adapt to local environments. Solutions such as IBM Access Connections [24] help by changing settings for default printers, network parameters, etc., as users move from one location to another.

Table 8 lists some of the benefits and drawbacks of the different solutions. Each solution comes with tradeoffs that are acceptable in certain environments. Approaches that separate the applications from the user interface require reliable low-latency networks and preinstalled client software, do not require any portable devices, and have fast suspend and resume times. based Client virtualization-based solutions that do not require the user to carry anything require high-bandwidth networks and preinstalled client software, have moderate suspend/resume times, and good interactive performance. However, they do not yet address processor architecture dependencies well. Approaches that require bulky devices to be carried are able to suspend and resume quickly and do not require networking.

Our approach is particularly well suited to environments and situations where connectivity is poor and the software state of environmental computers is unpredictable. The user is required to carry a device and suitable mechanisms have to be built to protect and recover from loss of the device. However, a significant advantage is that the *EnviroPCs* can be diskless, unmanaged devices. We leave no trace on the *EnviroPC* and therefore it is immediately available for other users. We also believe that our model can enable more users in developing countries to benefit from the computing revolution by requiring them to own a cheaper device instead of a full PC.

7 Conclusions

We have built a system that allows a user to walk up to a class of generic PCs and resume a suspended session by attaching a portable device. We provided detailed measurements using several types of portable *SoulPads* and *EnviroPC* configurations. The time to resume a session is in the order of two minutes and the time to suspend is in the order of thirty seconds. Application response time degradation for the SYSmark 2002 benchmark is around 40%. We believe that suspend/resume latencies and application performance are in the acceptable range for many users. Our approach incurs an increased resume time due to the Host OS auto-configuration needed to be independent of any installed software on the *EnviroPC*.

To our knowledge, the ability to resume suspended user sessions on standard PCs containing no software is unique to our approach. Issues surrounding the loss of the *SoulPad* are similar to ones for losing laptops, though one could argue that smaller devices are easier to lose. Backups can be done opportunistically to a server while connected to a network, or to local storage alternatives when networked services are not available.

We also reported several lessons learned from our experience. Processor architecture evolution needs to be addressed by virtual machine technologies for this type of migration to work across a broader class of machines. Software licensing terms need to be reexamined with the advent of VM-based mobility. Both these issues need to be addressed by OS, VMM, and application vendors.

Clearly, the use of virtual machines for migration of user environments is a promising approach. However, further work is needed before this vision can be realized in a broad set of computing environments. We hope that our work motivates the community to address some of the issues we have raised. New directions for combining *SoulPad* with portable music players, mobile phones, digital cameras, etc., should also be explored further.

8 Acknowledgements

We would like to thank the anonymous reviewers and our shepherd, M. Satyanarayanan, for comments that helped improve the paper. Srinivas Krishnamurti, Mendel Rosenblum and Michael Yang helped us obtain permission to publish benchmark results involving VMware Workstation 4. John Peterson provided a copy of SYSmark 2002 and helped us obtain permission to publish the results. SYSmark is a trademark of Business Applications Performance Corporation.

9 References

1. R. Baratto, S. Potter, G. Su and J. Nieh, *MobiDesk: Mobile Virtual Desktop Computing*, Proc. ACM MobiCom 2004.
2. P.M. Chen and B. D. Noble, *When Virtual is Better Than Real*, Proc. IEEE HotOS 2001.
3. F. Douglis and J. K. Ousterhout, *Transparent Process Migration: Design Alternatives and the Sprite Implementation,* Software - Practice and Experience, 21(8), 1991.
4. R. P. Goldberg, *Survey of Virtual Machine Research*, IEEE Computer, 7(6), 1974.
5. C. Helfrich, *private communication*, June 2004.
6. D. Husemann, C. Narayanaswami and M. Nidd, *Personal Mobile Hub*, Proc. IEEE International Symposium on Wearable Computers 2004.
7. M. Kozuch and M. Satyanarayanan, *Internet Suspend /Resume*, Proc. IEEE WMCSA 2002.
8. M. Kozuch et al., *Seamless Mobile Computing on Fixed Infrastructure*, IEEE Computer, 37(7), 2004.
9. M.J. Litzkow, *Remote Unix: Turning Idle Workstations into Cycle Servers,* Proc. Summer 1987 USENIX Conference.
10. D.A. Nichols, *Using Idle Workstations in a Shared Computing Environment*, Proc. ACM SOSP 1987.
11. T. Richardson, Q. Stafford-Fraser, K. R. Wood and A. Hopper, *Virtual Network Computing*, IEEE Internet Computing, 2(1), 1998.
12. T. Richardson, F. Bennett, G. Mapp and A. Hopper, *Teleporting in an X Window System Environment,* IEEE Personal Communications, 1(3), 1994.
13. C. P. Sapuntzakis et al., *Optimizing the Migration of Virtual Computers,* Proc. USENIX OSDI 2002.
14. M. Satyanarayanan, *The Evolution of Coda,* ACM TOCS, 20(2), 2002.
15. E. Solomita, J. Kempf and D. Duchamp, *XMOVE: A Pseudoserver for X Window Movement*, The X Resource, 11(1), 1994.
16. B. K. Schmidt, M. S. Lam and J. D. Northcutt, *The Interactive Performance of SLIM: a Stateless, Thin-Client Architecture*, Proc. ACM SOSP 1999.
17. M. Theimer, K. A. Lantz and D. R. Cheriton, *Preemptable Remote Execution Facilities for the V System,* Proc. ACM SOSP 1985.
18. N. Tolia, J. Harkes, M. Kozuch and M. Satyanarayanan, *Integrating Portable and Distributed Storage*, Proc. USENIX FAST 2004.
19. T. E. Truman, T. Pering, R. Doering and R. W. Broderson, *The InfoPad Multimedia Terminal: A Portable Device for Wireless Information Access,* IEEE Transactions on Computers, 47(10), 1998.
20. P. Vettiger et al., *The "Millipede" - Nanotechnology Entering Data Storage,* IEEE Transactions on Nanotechnology, 1(1), 2002.
21. R. Want et al, *The Personal Server: Changing the Way We Think about Ubiquitous Computing*, Proc. UbiComp 2002.
22. BAPCO, *An Overview of SYSmark 2002*, `http://www.bapco.com/techdocs/SYSmark2002Methodology.pdf`
23. Trusted Platform Module (TPM) specification, `https://www.trustedcomputinggroup.org/`
24. Please consult your favorite search engine.

Slingshot: Deploying Stateful Services in Wireless Hotspots

Ya-Yunn Su and Jason Flinn
Department of Electrical Engineering and Computer Science
University of Michigan

Abstract

Given a sufficiently good network connection, even a handheld computer can run extremely resource-intensive applications by executing the demanding portions on a remote server. At first glance, the increasingly ubiquitous deployment of wireless hotspots seems to offer the connectivity needed for remote execution. However, we show that the backhaul connection from the hotspot to the Internet can be a prohibitive bottleneck for interactive applications. To eliminate this bottleneck, we propose a new architecture, called Slingshot, that replicates remote application state on surrogate computers co-located with wireless access points. The first-class replica of each application executes on a remote server owned by the handheld user; this offers a safe haven for application state in the event of surrogate failure. Slingshot deploys second-class replicas on nearby surrogates to improve application response time. A proxy on the handheld broadcasts each application request to all replicas and returns the first response it receives. We have modified a speech recognizer and a remote desktop to use Slingshot. Our results show that these applications execute 2.6 times faster with Slingshot than with remote execution.

1 Introduction

Creating applications that execute on small, mobile computers is challenging. On one hand, the size and weight constraints of handheld and similar computers limit their processing power, battery capacity, and memory size. On the other hand, user's appetites are driven by the applications that run on desktops; these often require more resources than a handheld provides. A solution to this challenge is remote execution using wireless networks to access compute servers; this combines the mobility of handhelds and the processing power of desktops.

Although Internet connectivity is increasingly ubiquitous due to widespread deployment of wireless hotspots, the backhaul connections between hotspots and the Internet are communication bottlenecks. The uplink bandwidth from a wireless hotspot can be quite limited (e.g. 1.5 Mb/s for a T1 line). Further, this bandwidth must be shared by all hotspot users. The network round-trip time between a hotspot and a remote server may be large due to the use of firewalls and other middleboxes, as well as the vagaries of Internet routing. For interactive applications such as speech recognition and remote desktops, the combination of high latency and low bandwidth is prohibitive; mobile users cannot achieve acceptable response times when communicating with remote servers.

In this paper, we describe Slingshot, a new architecture for deploying mobile services at wireless hotspots. Slingshot replicates applications on *surrogate computers* [1] located at hotspots. A first-class replica of each application executes on a remote server owned by the mobile user. Slingshot instantiates second-class replicas on surrogates at or near the hotspot where the user is located. A proxy running on a handheld broadcasts each application request to all replicas; it returns the first response it receives to the application. Second-class replicas improve interactive response time since they are reachable through low-latency, high-bandwidth connections (e.g. 54 Mb/s for 802.11g). At the same time, the first-class replica is a trusted repository for application state that is not lost in the event of surrogate failure.

Slingshot also simplifies surrogate management. It uses virtual machine encapsulation to eliminate the need to install application-specific code on surrogates. Further, replication prevents the loss of application state when a surrogate crashes or even permanently fails. The performance impact of surrogate failure is mitigated by other replicas, which continue to service client requests.

The harnessing of surrogate computation is a multifaceted problem with many challenges. This paper addresses several of these challenges, including improving interactive response time, hiding the perceived cost of migration, recovering from surrogate failure, and simplifying surrogate management. It also presents concrete results that measure the potential benefit of surrogate computation for stateless and stateful applications. Other challenges remain to be addressed. Slingshot does not yet address privacy concerns, provide protocols for

secure replica management, manage surrogate load, or decide when to instantiate and destroy replicas.

We have implemented two Slingshot services: a speech recognizer and a remote desktop. Our results show that instantiating a second-class replica on a surrogate lets these applications run 2.6 times faster. Our results also show that replication lets Slingshot move services between surrogates with little user-perceived latency and recover gracefully from surrogate failure.

2 Design principles

We begin by discussing the three principles we followed in the design of Slingshot.

2.1 Location, location, location

Server location can be critical to the performance of remote execution. Consider a handheld connected to the Internet at a wireless hotspot. If the handheld executes code on a remote server, its network communication not only passes through the wireless medium; it also traverses the hotspot's backhaul connection and the wide-area Internet link. In a typical hotspot, the backhaul connection is the bottleneck. For instance, the nominal bandwidth of a 802.11g network (54 Mb/s) is more than an order of magnitude greater than that of a T1 connection. If the handheld could instead execute code on a server located at the hotspot, it could avoid the communication delay associated with the bottleneck link. For interactive applications that require sub-second response time, server location can make the difference between acceptable and unacceptable performance.

Network latency is also a concern. A server that is nearby in physical distance can often be quite distant in network topology due to the vagaries of Internet routing. Firewalls, VPNs, and NAT middleboxes add additional latency when connections cross administrative boundaries. For mobile users, a journey of only a few hundred yards can dramatically increase the round-trip time for communication with a remote server. In contrast, a server located at the current hotspot is only a network hop away.

2.2 Replicate rather than migrate

The desire to locate services near mobile users implies that services need to move over time. When a handheld user moves to a new location, a surrogate at the new hotspot will often offer better response time than a surrogate at the previous hotspot.

What is the best method to move functionality? One option is migration: suspend the application on the previous surrogate, transmit its state to the new surrogate, and resume it there. This approach has a substantial drawback: the application is unavailable while it is migrating. Slingshot uses an alternative strategy that instantiates multiple replicas of each service. While a new replica is being instantiated, existing replicas continue to serve the user.

Slingshot replication is a form of primary-backup fault tolerance; i.e. it tolerates the failure of any number of surrogates. For each application, Slingshot creates a first-class replica on a reliable server known to the mobile user—this server is referred to as the *home server*. Slingshot ensures that all application state can be reconstructed from information stored on the client and the home server. This allows all state on a surrogate to be regarded as soft state. Even if all surrogates crash, Slingshot continues to service requests using the first-class replica on the home server. In contrast, a naive approach that migrates applications between surrogates might lose state when a surrogate fails.

We note that Slingshot handles both *stateful* and *stateless* applications. The result of a remote operation for a stateful application depends upon the operations that have previously executed. Slingshot assumes that applications are deterministic; i.e. that given two replicas in the same initial state, an identical sequence of operations sent to each replica will produce identical results. As we discuss in Section 4.2, Slingshot adopts an approach similar to that of Rodrigues' BASE [24] in eliminating nondeterminism with wrapper code. Slingshot instantiates a new replica by checkpointing the first-class replica, shipping its volatile state to a surrogate, and replaying any operations that occurred after the checkpoint.

Instantiation of a new replica takes several minutes since the volatile state must travel through the bandwidth-constrained backhaul connection. However, existing replicas mitigate the perceived performance impact. Until the new replica is instantiated, existing replicas service application requests.

2.3 Ease of maintenance

We see the business case for deploying a surrogate as being similar to that of deploying a wireless access point. Desktop computers (without monitors) cost only a few hundred dollars today, not much more than an access point. Further, our results show that surrogates can provide significant value-add to wireless customers in terms of improved interactive performance.

However, surrogates must be easy to manage if they are to be widely deployed. Since we envision surrogates at hotspots in airport lounges, coffee shops, and bookstores, they must require little to no supervision. They should be appliances that require little configuration; most problems should be fixable with a reboot.

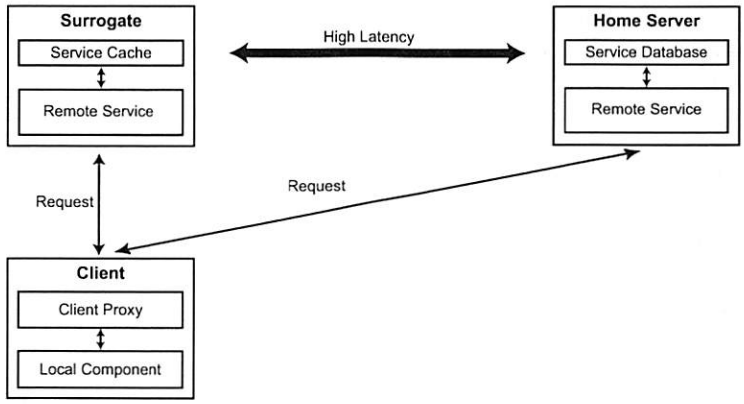


Figure 1. Slingshot architecture

To make surrogates easy to manage, Slingshot:

- **minimizes the surrogate computing base.** Each replica runs within its own virtual machine, which encapsulates all-application specific state such as a guest OS, shared libraries, executables, and data files. The surrogate computing base consists of only the host operating system (Linux), the virtual machine monitor (VMware), and Slingshot. No configuration or setup is needed to enable a surrogate to run new applications—each VM is self-contained.
- **uses a heavyweight virtual machine**. While para-virtualization and other lightweight approaches to virtualization offer scalability and performance benefits [4, 23, 29], they also restrict the type of applications that can run within a VM. In contrast, by using a heavyweight VMM (VMware), Slingshot runs the two applications described in Section 4 without modifying source code, even though their guest OS (Windows XP) differs substantially from the surrogate host OS (Linux).
- **places no hard state on surrogates.** Because surrogates have only soft state, a reboot does not lead to incorrect application behavior or data loss. If a surrogate crashes or is rebooted, the only impact a user sees is that performance declines to the level that would have been available had the surrogate never been present.

3 Slingshot implementation

3.1 Overview

Figure 1 shows an overview of Slingshot. For simplicity of exposition, this figure assumes that the mobile client is executing a single application and that a single surrogate is being used. In practice, we expect a Slingshot user to run only one or two applications concurrently, with each service replicated two or three times.

Each Slingshot application is partitioned into a *local component* that runs on the mobile client and a *remote service* that is replicated on the home server and surrogates. Ideally, we partition the application so that resource-intensive functionality executes as part of the remote service; the local component typically contains only the user interface. This partitioning enables demanding applications to run on clients such as handhelds that are highly portable but also resource-impoverished. The applications that we have studied so far (speech recognition and remote desktops) already had client-server partitionings that fit this model. For some applications, the best partitioning may not be immediately clear—in these cases, we could leverage prior work [2, 12, 18] to choose a partition that fits our model.

In Figure 1, a first-class replica executes on the home server and a second-class replica executes on the surrogate. The home server, described in Section 3.2, is a well-maintained server under the administrative control of the user, e.g. the user's desktop or a shared server maintained by the user's IT department. In contrast, surrogate computers, described in section 3.3, are co-located with wireless access points. They are administered by third parties and are not assumed to be reliable.

Slingshot creates the first-class replica when the user starts the application—this replica is required for execution of stateful services. As the application runs, Slingshot dynamically instantiates one or more second-class replicas on nearby surrogates. These replicas improve interactive performance because they are located closer to the user and respond faster than the first-class replica on the distant home server. If no second-class replicas are instantiated, Slingshot's behavior is identical to that of remote execution.

Each replica executes within its own virtual machine. Replica state consists of the *persistent state*, or disk im-

age of the virtual machine, and the *volatile state*, which includes its memory image and registers. To handle persistent state, we use the Fauxide and Vulpes modules developed by Intel Research's Internet Suspend/Resume (ISR) project [20]. These modules intercept VMware disk I/O requests. On the home server, we redirect these requests to a *service database* that stores the disk blocks of every remote service. On a surrogate, VMware reads are first directed to a *service cache*—if the block is not found in the cache, it is fetched from the service database on the home server.

The client proxy is responsible for locating surrogates, instantiating second-class replicas, and managing communication with all replicas. It presents the local component with the illusion that it is using a single remote service by broadcasting each request to all replicas and forwarding the first reply it receives to the local component. Later replies from other replicas are checked for consistency, as described in Section 3.4.

If a mobile computer has a high-capacity storage device such as a flash card or a microdrive, Slingshot reduces the time to instantiate replicas by storing checkpoints on the mobile computer. As described in Section 3.6, the client logs all operations that occur after the checkpoint and replays them to bring a new surrogate up-to-date.

3.2 The home server

A Slingshot user defines a single, well-known home server that stores and executes the first-class replicas for all of her remote services. Each service is uniquely identified by a *serviceid* string assigned by the user when the service is created. The service database on the home server manages the persistent and volatile state associated with each service. The *director* instantiates and terminates first-class replicas. We describe these components in the next two sections. The home server also runs the VMware virtual machine monitor. Each Slingshot service runs within a separate VM that is dynamically instantiated when a user starts its associated application on the client.

3.2.1 The service database

The home server stores the state of every service under its purview in its service database. Previous research in virtual machine migration by Sapuntzakis [25] and Tolia [26] has shown that content-addressable storage is highly effective in reducing the storage costs of virtual machine disk images. We adopt their approach by dividing the disk image of each virtual machine into 4 KB chunks and indexing each chunk by its SHA-1 hash value. As shown in Figure 2, each service has a *chunk table* that maps the chunks in its virtual disk image to the SHA-1 hash of the data stored at each location.

The service database assumes that any two blocks that hash to the same value are identical. It maintains a hash table of the SHA-1 values of all chunks that it currently stores. When it receives a request to store a new chunk whose SHA-1 value matches that of a chunk it already has stored, it increments a reference count on the existing chunk. This method of eliminating duplicate storage has been shown to substantially reduce disk usage [10] due to similarities between the disk state of different computers. We expect such similarities to be common in our environment, since a single user may create many remote services from a generic OS image. For example, we created the speech recognizer and VNC services discussed in Section 4 from the same Windows XP image.

As shown in Figure 2, when a first-class replica reads a block from its virtual disk, the Fauxide/Vulpes ISR modules intercept the request and pass the associated logical block number to the service database. The database looks up the block number in the service's chunk table to determine the SHA-1 value of the chunk stored at that location. It then looks up the SHA-1 value in the hash table to find the location of the chunk in the database.

Requests that modify blocks follow a similar path. The database locates the chunk in the service's chunk table. It then indexes on the old SHA-1 value and decrements the reference count associated with the chunk in its hash table. If the reference count drops to zero, it deletes the chunk. The service database next looks up the new SHA-1 value of the modified block in its hash table. If the modified chunk is a duplicate of an existing chunk, the service database increments the reference count of the existing chunk. Otherwise, it stores the chunk and adds its SHA-1 value to its hash table.

Since the volatile state is likely unique to each service, content-addressable storage offers little benefit. Thus, the service database stores the volatile state of each remote service in a file named by its serviceid.

3.2.2 The home server director

When a mobile user starts a Slingshot application, the client proxy asks the director on the home server to instantiate the first-class replica. The director uses the serviceid provided by the client proxy to retrieve the volatile state from the service database. It starts a VMware process, resumes the virtual machine with the volatile state, and replies to the client proxy. The persistent state is retrieved on demand from the service database as the first-class replica executes.

When the user terminates the application, the client proxy tells the director to halt the replica. The director suspends the virtual machine, which causes VMware to write its volatile state to disk. It then terminates the virtual machine.

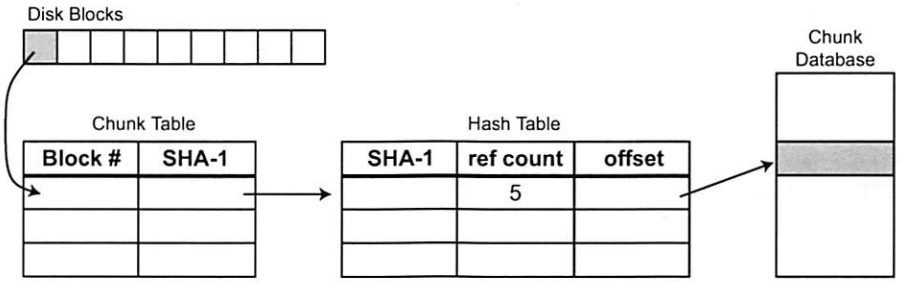


Figure 2. Reading a chunk from the service database

The volatile state is large (e.g. 128 MB) because it contains the entire memory image of the virtual machine. We use Waldspurger's *ballooning* technique [28] to reduce its size. When we create a new service, we place a script in the guest OS that allocates a large amount of highly-compressible (e.g. almost entirely zeroed out) memory pages. When VMware suspends the virtual machine, this script runs to force unused memory pages to disk and replace them with more compressible pages. The director compresses the volatile state with gzip before storing it in the service database—this reduces storage and network costs.

3.3 Surrogates

The surrogate architecture is similar to that of the home server, except that we replace the service database with the service cache described in Section 3.3.2. The director also plays a slightly different role on a surrogate.

3.3.1 The surrogate director

The surrogate director currently accepts connections from any client that wishes to instantiate a second-class replica. Potentially, the director could enforce access-control policies similar to those enforced by access points today. The client proxy passes the director the IP address of its home server and the serviceid of the remote service it wishes to instantiate. The director contacts the home server and requests the volatile state and chunk table for the requested service.

Usually, the home server is already executing the first-class replica of the service in question. For a stateful service, this means that Slingshot must generate a coherent checkpoint that represents the current execution state of that replica. The home server creates this checkpoint by suspending and resuming the virtual machine containing the replica; this causes a new volatile state and chunk table to be written to the service database.

The home servers ships copies of the volatile state and chunk table to the surrogate. Even after compression, this information is quite large (e.g. 32 MB for a VNC service)—thus, it can take several minutes to transfer. Next, the director starts a new virtual machine and resumes it using the volatile state. As the replica executes, its disk I/O is intercepted by the ISR modules and redirected to the service cache described below.

When the client disconnects from the surrogate, the director terminates the virtual machine. Since surrogate replicas are second-class, the service state is logically discarded at this point. However, persistent state chunks remain in the service cache until they are evicted due to storage limitations. This improves response time if the service is later restarted on the surrogate.

3.3.2 The service cache

The service cache is a content-addressable store of data chunks. As with the service database, each chunk is indexed by its SHA-1 hash value, and storage of duplicate chunks is eliminated. This lets users benefit from similarities among the disk images of their replicas. For instance, two people using Windows-based services may have similar disk images. Chunks cached by one user can be used by the other.

When the service cache receives a request to read a chunk, it first tries to service it locally. If the chunk is not cached, it asks the service database associated with the replica for the chunk.

A subtle problem occurs because Slingshot enforces determinism at the application level. There is no guarantee that two different replicas will write the same data to the same location on disk. A naive implementation might ask the database for an uncached chunk, only to find that it had been over-written by a store performed by the first-class replica. We therefore need to ensure that the service database keeps all chunks that might potentially be requested by second-class replicas.

Slingshot uses a copy-on-write approach for stateful services. When a surrogate starts a second-class replica, the database copies the service's chunk table—the new copy increments the reference count for each of its entries. When the second-class replica is terminated, the database deletes its chunk table and decrements the reference count for each entry. Thus, even if the first-class

replica modifies or deletes a chunk, that chunk is not deleted until after the second-class replica terminates.

A similar concern arises for chunks modified by the second-class replica. The modified chunks may not be written to the service database by the first-class replica due to non-determinism at the disk I/O level. The surrogate cache therefore pins modified chunks for the duration of a replica's execution—this ensures that they will never need to be fetched from the service database.

The surrogate cache uses an LRU eviction algorithm that exempts chunks that are currently pinned. Since chunks remain cached even after a service is terminated, it is likely that the chunks of a frequent visitor to a hotspot will remain cached between visits.

3.4 The client proxy

The client proxy is a stand-alone process responsible for surrogate discovery, instantiating and destroying replicas, and coordinating communication with each replica. It uses UPnP [22] to discover new surrogates in its surrounding network environment. Currently, the decision to instantiate a new second-class replica is a made by the user. In the future, we plan to add heuristics for monitoring network performance and automatically deciding when new replicas are needed.

On startup, the local component sends the client proxy its serviceid. The proxy immediately instantiates a first-class replica on the home server. It subsequently instantiates second-class replicas on nearby surrogates when requested by the user.

The client proxy maintains an *event log* of requests sent by the local application component. The client proxy spawns a thread for each replica; the thread sends logged events to the replica in the order they were received. Events may optionally have application-specific preconditions that must be satisfied before they can be sent to a replica. For instance, our VNC application specifies a precondition that ensures that the remote desktop is ready to accept each key stroke and mouse click event before that event is sent. Services that must process events sequentially to ensure determinism specify that the previous event must complete before an event is sent.

The client proxy records the replies received from each replica in the event log. When the first reply is received, it is returned to the local component. Later replies are compared with the first reply to ensure that the replicas are behaving deterministically. If the reply from a second-class replica differs significantly (as determined by an application-specific function) from the reply from the first-class replica, the second-class replica is terminated. Such divergence could be due to a bug in the wrapper code enforcing determinism, or it could be the result of a faulty or malicious surrogate. Note that the client proxy may already have received a reply that is later determined to be faulty. In this case, the application is notified via an upcall so that corrective action can be taken. This strategy is similar to those employed in the SUNDR file system [21] and in Brown's operator undo [7]. Alternatively, we could try to prevent malicious surrogate behavior using a trusted computing architecture [15].

3.5 Instantiating new replicas

In Figure 3, we show how Slingshot responds to a user moving between hotspots, assuming that a surrogate is located at each hotspot. The client proxy first asks the nearby surrogate to instantiate a replica. The surrogate requests the service state from the home server; the home server checkpoints the first-class replica and ships the compressed chunk table and volatile state to the surrogate. Note that the distant surrogate can process events for the client during checkpointing. This hides almost all delay associated with suspending and resuming the VM on the home server. The client proxy queues events for the first-class replica while it is being checkpointed and sends them after the replica resumes execution.

The new surrogate uses the checkpoint to resume the service within a new virtual machine. The client proxy brings the new replica up-to-date by replaying all events in the event log that were sent by the application after the checkpoint was created. Once the new replica is up-to-date, it improves interactive response time for the application by responding more swiftly to new events sent by the local component. At this point, the client proxy terminates the replica at the previous hotspot.

The benefit of replication is that the user sees little foreground performance impact due to the use of a new surrogate. After checkpointing, the first-class replica on the home server services requests while the new second-class replica is instantiated and brought up-to-date. In contrast, a naive migration approach would leave the service unavailable while state is being shipped—as we show in Section 5, application state can take several minutes to ship over limited backhaul connections. Although the first-class replica is unavailable while it is being checkpointed, that operation is relatively short (i.e. approximately 10 seconds). Even that cost can be masked if another second-class replica exists.

Slingshot performs two optimizations if a service is marked as stateless. It skips checkpointing the service on the home server (since its state is static). It also does not replay events (since the replica is up-to-date).

3.6 Leveraging mobile storage

Migration can be time consuming due to the need to ship state from the home server (step 4 in Figure 3). For a typ-

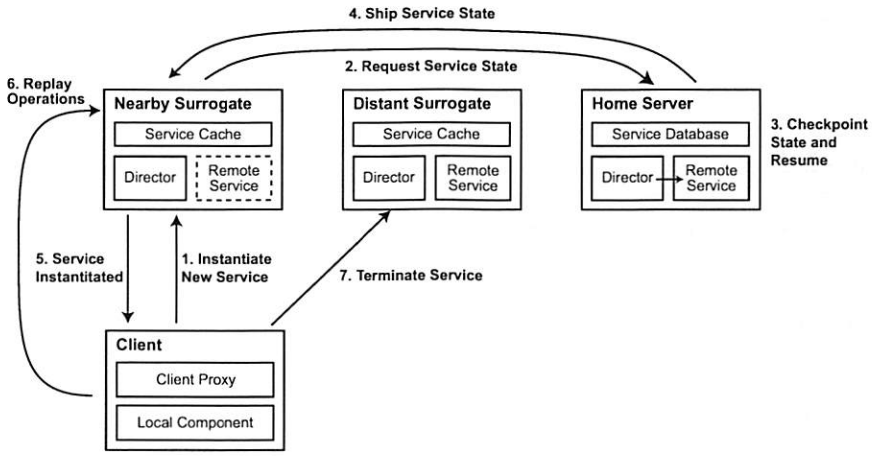


Figure 3. Instantiating a new replica

ical service, the size of the compressed volatile state and chunk table is 30–40 MB. If the home server is connected to the Internet via a DSL link with 256 Kb/s uplink bandwidth, it takes over 20 minutes to ship the state.

Given sufficient storage capacity, Slingshot reduces the time to ship state by storing checkpoints on the mobile computer. We observed that the service-specific event log can be used to roll forward replica state from *any* prior checkpoint, not just one that is created at the beginning of replica instantiation. Thus, by storing a checkpoint on a mobile computer and logging all events that occur after that checkpoint, Slingshot can instantiate a replica without shipping state from the home server. Instead, it ships the state from the mobile computer over the high-bandwidth wireless network at the hotspot. Most of this bandwidth should be unused since the capacity of the wireless network is typically much greater than that of the backhaul connection, yet most communication from computers located at the hotspot is with endpoints located outside the hotspot (and thus limited by the backhaul bandwidth).

When a user returns to her home server, she can tell Slingshot to create new checkpoints of her applications on a high-capacity storage device such as a microdrive. Each checkpoint contains the volatile and persistent state. The volatile state and chunk table are stored in separate files; the chunks that comprise the persistent state are stored in a content-addressable cache on the mobile computer. The event log is empty when the user creates a new snapshot. As the application is used on the road, Slingshot appends each request to the log. This enables Slingshot to instantiate a new replica of a stateful service by first restoring the checkpoint represented by the volatile state, and then deterministically replaying the event log. For stateless services, Slingshot neither records nor replays an event log.

When a new replica is instantiated on a nearby surrogate, the mobile computer tries to find a checkpoint on its local storage device. If a checkpoint is found, the mobile computer ships the volatile state, chunk table, and hash table for its local chunk cache to the surrogate. One reason that we transmit the chunk table and the hash table to the surrogate is that the surrogate can usually maintain this information in memory, whereas a resource-constrained mobile computer cannot. When a disk I/O request misses in the service cache, the surrogate fetches the chunk from the mobile computer if it is available there; otherwise, the chunk is fetched from the home server.

As operations accumulate, so does the time to bring a new second-class replica up-to-date. This means that there exists a break-even point where it takes less time to create a new checkpoint from the first-class replica on the home server and download it over the Internet than it takes to instantiate a replica from client storage.

4 Slingshot applications

We have adapted the IBM ViaVoice speech recognizer and the VNC remote desktop to use Slingshot. Due to Slingshot's use of virtual machine encapsulation, we did not need to modify the source code of either application. All Slingshot-specific functionality is performed within proxies that intercept and redirect network traffic.

4.1 Speech recognition

We chose speech recognition as our first service because of its natural application to handheld computers. We used IBM ViaVoice in our work. We created a server-side proxy that accepts audio input from a remote client and passes it to ViaVoice through that application's API. ViaVoice returns a text string which the proxy sends to the client. ViaVoice and our server run on a Windows XP guest OS executing within a VMware virtual ma-

chine. The local component of this application displays the speech recognition output.

We chose to implement speech recognition as a stateless service. One can certainly make a reasonable argument that speech recognition should be a stateful service in order to allow a user to train the recognizer. However, we wanted to explore the optimizations that Slingshot could provide for stateless services.

4.2 Virtual desktop

VNC allows users to view and interact with another computer from a mobile device. In the case of Slingshot, the remote desktop is a Windows XP guest OS executing within a VMware virtual machine. This allows users to remotely execute any Windows application from their handhelds. This is clearly a stateful service; i.e., a user who edits a Word document expects the document to exist when the service is next instantiated.

Adapting VNC to Slingshot presented interesting challenges. First, the VNC server sends display updates to the client in a non-deterministic fashion. When pixels on the screen change, it reports the new values to the client in a series of updates. Two identical replicas may communicate the same change with a different sequence of updates. The resulting screen image at the end of the updates is identical but the intermediary states may not be equivalent. A second challenge is that some applications are inherently non-deterministic. One annoying example is the Windows system clock; two surrogates can send different updates because their clocks differ.

We noted that some non-determinism is unlikely to be relevant to the user (e.g. a slightly different clock value). Unfortunately, other non-determinism affects correct execution. For example, a key stroke or mouse click is often dependent upon the window state. If a user opens a text editor and enters some text, the key strokes must be sent to each replica only after the editor has opened on that replica. If this is not done, the key strokes will be sent to another application. To solve this problem, we associate a precondition with each input event. When the user executes the event, we log the state of the window on the client to which that event was delivered. When replaying the event on a server, we require that the window be in an identical state before the event is delivered. Since each event is associated with a screen coordinate, we check state equality by comparing the surrounding pixel values of the original execution and the background execution. In the above example, this strategy causes Slingshot to wait until the editor is displayed before it delivers the text entry events.

A second issue with VNC is that its non-determinism prevents us from mixing updates from different replicas. We designate the best-performing replica as the foreground replica and the remainder as background replicas. Only events from the foreground replica are delivered to the client. If performance changes, we quiesce the replicas before choosing a new foreground replica. Two replicas are quiesced by ensuring that the same events have been delivered to each, and by requesting a full-screen update from the new foreground replica to eliminate transition artifacts. New events are logged while quiescing replicas. Note that the foreground replica is rarely the first-class replica since nearby surrogates provide better performance in the common case.

We were encouraged that VNC can fit within the Slingshot model, since its behavior is relatively non-deterministic. Based on this result, we suspect that application-specific wrappers can be used to enforce determinism for many applications. For those applications where this approach proves infeasible, we could use a VMM that enforces determinism at the ISA level as is done in Hypervisor [6] and ReVirt [11].

5 Evaluation

Our evaluation answers the following questions:

- How much do surrogates improve interactive response time?
- What is the perceived performance impact of instantiating a new replica?
- How much does the use of mobile storage reduce replica instantiation time?

5.1 Methodology

The client platform in our evaluation is an iPAQ 3970 handheld running the Linux 2.4.19-rmk6 kernel. The handheld has an XScale-PXA250 processor, 64 MB of DRAM, and 48 MB of flash. It uses a 11 Mb/s Cisco 350 802.11b PCMCIA card for network communication and a 4 GB Hitachi microdrive for bulk storage. Unless otherwise noted, the home server and surrogates are Dell Precision 350 desktops with a 3.06 GHz Pentium 4 processor running RedHat Enterprise Linux version 3.

We use a Cisco 350 802.11b wireless access point. We emulate the topology in Figure 4 by connecting all computers and the access point to a Dell desktop running the NISTNet [8] network emulator. This topology emulates a scenario where the handheld client is located at a wireless hotspot equipped with a surrogate computer. Hotspots are connected to the Internet through T1 connections with 1.5 Mb/s uplink and downlink bandwidth. A distant surrogate at another hotspot is accessible with latency of 15 ms. The home server is connected through

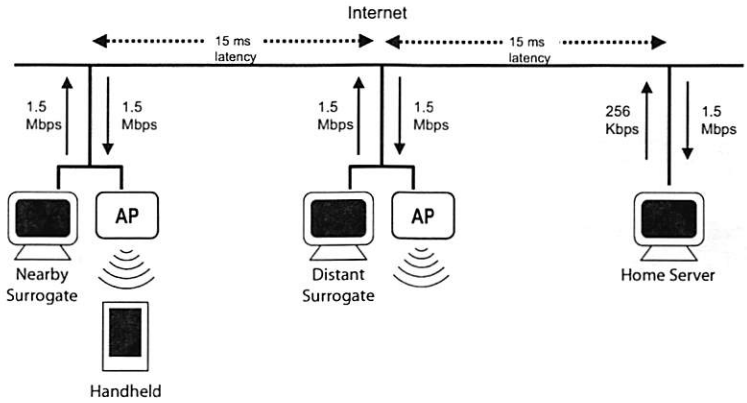


Figure 4. Network topology used in evaluation

an emulated DSL connection—the latency between the handheld's hotspot and the home server is 30 ms.

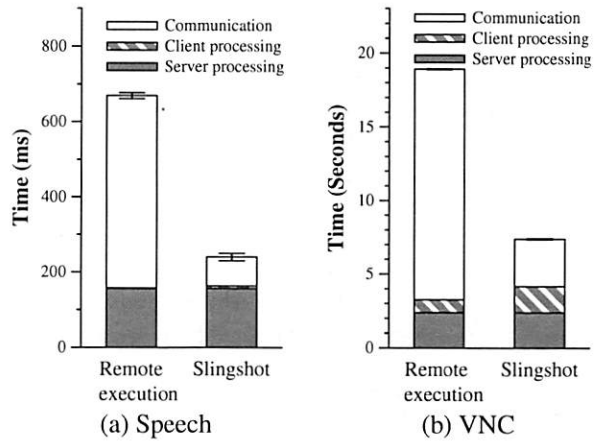
We execute the IBM ViaVoice speech recognizer as a stateless service, and VNC as a stateful service. For repeatability, the local component of each application executes a fixed, periodic workload. For speech, each iteration of the workload recognizes a phrase and pauses three seconds before beginning the next iteration. For VNC, each iteration opens Microsoft Word, inserts text at the beginning of a document, saves the document, closes Word, and pauses ten seconds before the next iteration begins. The client uses the same heuristics described in Section 4.2 to wait until Word opens before inserting text, and to wait until the window is fully closed before beginning the 10 second pause between iterations.

Each service runs within a separate VM configured with 128 MB of memory and 4 GB of local storage. We create each service from a vanilla Windows XP installation. We install the ballooning script described in Section 3.2.2 and the application comprising the remote service. We start the application so that it is ready to receive incoming connections, then suspend the VM. We repeat each experiment three times and report mean results over all iterations during the three trials. Figures 12 and 13 summarize all results described in this section.

5.2 Benefit of Slingshot

We first measured the benefit of using Slingshot for our two applications. The left bar in each data set in Figure 5 shows the average time to perform an iteration of the workload when the service is remotely executed on the home server. The right bar shows the average time using Slingshot when a second-class replica executes on the nearby surrogate. We let each application run for several iterations before measuring performance; this eliminates startup transients and shows steady-state performance.

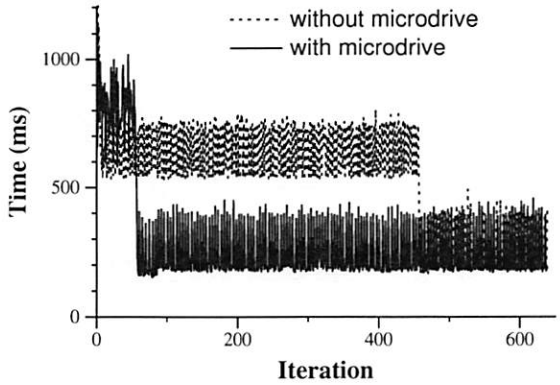
Both the stateless speech service and the stateful VNC service execute 2.6 times faster with Slingshot than with remote execution. The shadings within each bar show the time consumed by server processing, client processing, and communication. For speech, Slingshot increases client processing time since it manages multiple network connections and aggregates responses. For VNC, it also logs requests and responses to local storage. Slingshot's performance benefit comes from reducing the time the application blocks on network communication.



This graph compares the average time to execute the speech and VNC workloads when using remote execution and when using Slingshot. Each bar shows mean response time—the error bars are 90% confidence intervals.

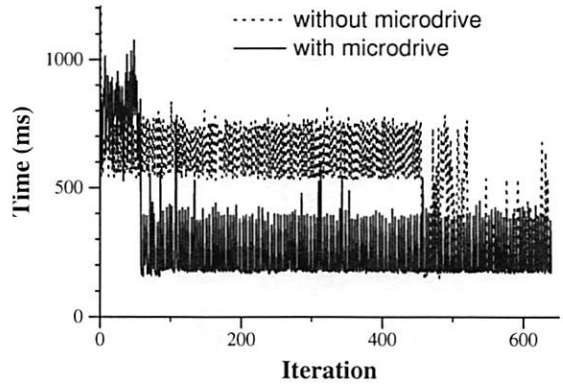
Figure 5. Benefit of using Slingshot

Remote execution performance is affected by both high latency and limited bandwidth. For speech, a back-of-the-envelope calculation shows that 229 ms are required to transfer the 44 KB utterance through the bottleneck 1.5 Mb/s T1 link at the hotspot. Further, since communication is intermittent, TCP slow start causes several 60 ms round-trip delays during transmission. Thus, the remote execution results include 511 ms of network communication time. In contrast, Slingshot uses only 77 ms for network communication.



This graph shows how response time changes during the instantiation of a speech replica on the nearby surrogate. All chunks are in the service cache prior to each experiment.

Figure 6. Speech replication with warm cache



This graph shows how response time changes during the instantiation of a speech replica on the nearby surrogate. No chunks are in the service cache prior to each experiment.

Figure 7. Speech replication with cold cache

Latency impacts VNC performance more than bandwidth. Because the client waits for remote actions such as button clicks and key presses to complete before initiating the next action, there are many round-trip delays during the VNC workload. In addition, client polling in VNC leads to more round-trip delays than are strictly necessary. For this workload, remote execution on the home server requires 15.6 seconds for network communication, while Slingshot requires only 3.2 seconds.

5.3 Stateless service replication

We next examined the impact of instantiating stateless second-class replicas. In this experiment, a user with a first-class replica running on the home server arrives at the hotspot on the left in Figure 4 and decides to instantiate a replica on the surrogate there. For repeatability, we do not measure the latency of UPnP service discovery. We examine behavior when the service cache is cold (i.e. no chunks are initially cached) and warm (i.e. all chunks are initially cached). The warm cache scenario is most likely if the user has recently visited the hotspot; the cold cache scenario is the worst cache state possible.

We first ran three trials without a microdrive attached to the iPAQ. Since the handheld has limited storage, the service state must be loaded from the home server as described in Section 3.5. We then ran three trials with the microdrive; in this case, the state of the speech service is loaded from the iPAQ as described in Section 3.6. Figures 6 and 7 show results for representative trials with a warm and cold cache, respectively.

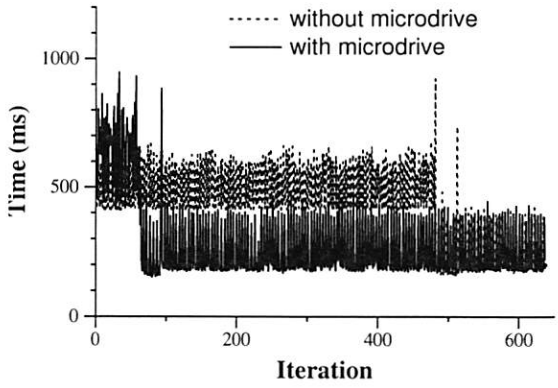
In Figure 6, the sharp drop in response time for both lines is a result of the completion of replica instantiation. Before the replica is instantiated, speech requests must be serviced by the distant home server; after instantiation, the new second-class replica provides quicker response time. Without the microdrive, it takes 28:06 minutes to ship the service state from the home server. However, replica instantiation exhibits only a minimal impact on application performance—average response time during replication is only 2% greater than response time with remote execution on the home server.

When the replica is instantiated from state stored on the client's microdrive, the new second-class replica is instantiated in only 3:35 minutes (7.8 times faster). However, the performance impact of replica instantiation is more substantial: average response time increases by 20% compared to remote execution. Shipping a large amount of data over the wireless network causes queuing delays at the access point and on the handheld that adversely affect application performance. Currently, we are investigating whether traffic prioritization can minimize the impact of replication on foreground traffic.

The cold cache scenario in Figure 7 exhibits a less clear difference in performance before and after replication completes. After the new replica is instantiated, it fetches chunks of its persistent state on demand from the home server or iPAQ; this occasionally delays its responses. Note that the first-class replica on the home server mitigates the performance impact—if the second-class replica is substantially delayed by fetching state, the first-class replica responds faster.

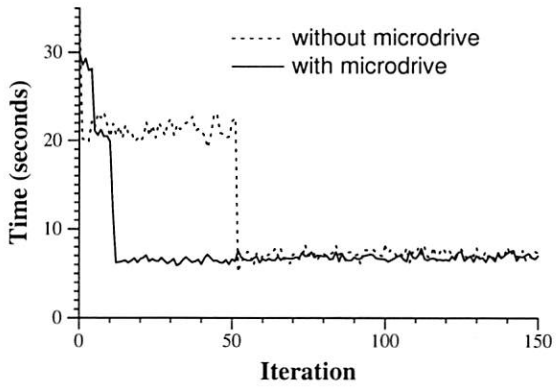
5.4 Instantiation of another stateless replica

We next examined a scenario in which the user of the speech service moves from one wireless hotspot to another. This experiment begins with the user located at the middle hotspot in Figure 4. A second-class replica is running on the surrogate at that hotspot and a first-class replica is running on the home server. At the beginning of the experiment, the user moves to the left hotspot and decides to instantiate another replica on the surrogate at that hotspot. While this new replica is being created,



This graph shows how response time changes during the instantiation of a speech replica on the nearby surrogate while another replica executes on the distant surrogate. All chunks are in the the service caches prior to each experiment.

Figure 8. Speech: Moving to a new hotspot



This graph shows how response time changes during the instantiation of a VNC replica on the nearby surrogate. All chunks are in the service cache prior to each experiment.

Figure 9. VNC replication with warm cache

both the second-class replica on the distant surrogate and the first-class replica on the home server service application requests. As soon as the new replica is instantiated, Slingshot terminates the replica on the distant surrogate. Since we did not have three identical servers with which to run this experiment, the home server is a slightly less-powerful Dell Optiplex 370 with 2.8 GHz Pentium 4 processor running RedHat 9.

Figure 8 shows how the average time to perform an iteration of the speech recognition workload varies during this experiment—we show only warm cache results here. Compared to the previous experiment, the time to instantiate a new replica is relatively unchanged. However, response time during replication is improved because the existing second-class replica responds faster to requests than the replica on the home server. Without the microdrive, application response time is reduced by 23% compared to remote execution; with the microdrive, application response time is reduced by 2%. These results show that a surrogate can still provide significant benefit even when not located at the user's current hotspot.

5.5 Stateful service replication

We next repeated the experiment in Section 5.3 for the stateful VNC service. Prior to the experiment, we perform 30 iterations of the VNC workload. We then begin the experiment by instantiating a replica on the nearby surrogate. Figures 9 and 10 show results from the warm and cold cache scenarios, respectively.

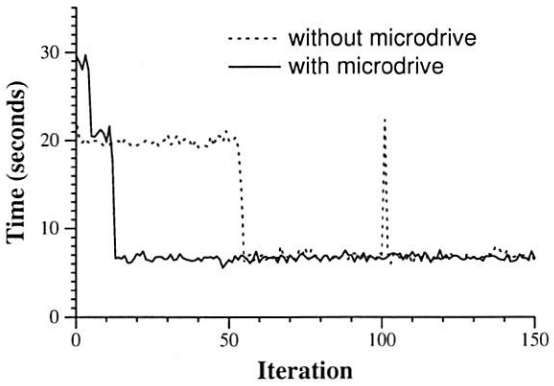
Without the microdrive and with a warm service cache, Slingshot takes 4 seconds to checkpoint the VNC service—this is reflected in the higher response time for the first iteration. Slingshot takes 22:42 minutes to ship the checkpoint to the surrogate and 5:02 to replay the logged operations. During replication, average response time is 20% higher than when using remote execution on the home server. This increase is due to the background traffic associated with shipping state from the home server interfering with the latency-sensitive foreground traffic of VNC.

In contrast to the prior results for the speech service, the VNC results show little difference between the warm and cold cache scenarios. In particular, VNC performance markedly improves in the cold cache scenario as soon as the client starts using the second-class replica. Most of the chunks needed by the service are read from the service database during the replay of logged operations.

When the handheld stores a VNC service checkpoint on its microdrive, Slingshot takes 3:19 minutes to ship the state from the client, and 3:18 minutes to replay the log. These two phases are clearly visible in the "with microdrive" line in Figure 9, where response time degrades by 52% compared to remote execution while state is being shipped, and by 9% while the log is replayed. Note that the log replay with the microdrive includes the 30 iterations that occurred prior to the experiment. Since the microdrive checkpoint is taken when the user leaves home, all logged operations after that point must be replayed. However, since shipping state takes less time with the microdrive, the user generates fewer logged operations during migration. Overall, Slingshot instantiates the replica over 4 times faster when a checkpoint exists on the microdrive.

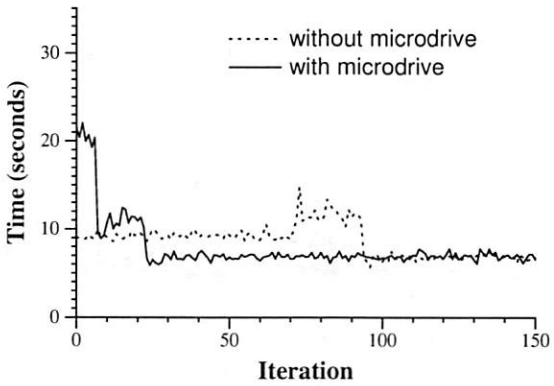
5.6 Instantiation of another stateful replica

We also repeated the experiment in Section 5.4 for VNC. Prior to the experiment, we create a second-class replica of the VNC service on the distant surrogate and a first-class replica on the home server. We then execute 30 iterations of the VNC workload. The experiment begins



This graph shows how response time changes during the instantiation of a VNC replica on the nearby surrogate. No chunks are in the surrogate cache prior to each experiment.

Figure 10. VNC replication with cold cache



This graph shows how response time changes during the instantiation of a VNC replica on the nearby surrogate while another replica executes on the distant surrogate. All chunks are in the the service caches prior to each experiment.

Figure 11. VNC: Moving to a new hotspot

when we start to instantiate another second-class replica on the nearby surrogate.

As shown in Figure 11, the presence of another second-class replica on the distant surrogate substantially improves performance during replication. Compared to remote execution, Slingshot provides VNC response times almost twice as fast without the microdrive, and 70% faster when state is fetched from client storage.

6 Related work

To the best of our knowledge, Slingshot is the first system to dynamically instantiate replicas of stateful applications in order to improve the performance of small, resource-poor mobile computers. Our work draws upon several areas of prior work, including virtual machine and process migration, cyber foraging, fault-tolerant computing, and remote execution.

After Chen and Noble [9] first suggested that virtual machine migration could be an effective mechanism for process migration, several research groups have built working prototypes. Our research focus is not on the migration mechanism itself, but rather on how it can be used to service the needs of small, mobile clients. We use the Fauxide and Vulpes components from Intel's Internet Suspend/Resume [20] to intercept disk I/O requests made by virtual machines. The difference between ISR and Slingshot is that ISR executes a user's computing environment on a single terminal at a time. In contrast, Slingshot decomposes a user's environment into distinct services and replicates services on multiple computers. Slingshot hides the perceived latency of migration and surrogate failures, while letting a user execute applications anywhere a wireless connection exists.

Sapuntzakis [25] uses virtual machine migration, but focuses on users who compute at fixed locations, rather than the mobile users that Slingshot targets. Slingshot uses several of the optimizations suggested by Sapuntzakis, including ballooning and content-addressable storage. These optimizations have also been suggested by Waldspurger [28] and Tolia [27], respectively.

Baratto's MobiDesk [3] is similar to our VNC application in that it virtualizes a remote desktop for mobile clients. However, MobiDesk migrates its desktop service between well-connected machines in a cluster in order to minimize downtime during server maintenance or upgrades. Slingshot uses replication rather than migration, and utilizes the computational resources of surrogates located at wireless hotspots. A MobiDesk-like cluster could serve as the ideal home server for Slingshot applications. Conversely, although Baratto shows considerable improvement over VNC in remote display performance, his results indicate that network latency still degrades interactive performance. Thus, surrogates could improve MobiDesk performance for mobile clients.

Slingshot's replication strategy is a form of primary-backup fault tolerance in that the replica on the home server allows the system to tolerate a fail-stop failure of any number of second-class replicas. Our approach is most reminiscent of Hypervisor [6], which used deterministic replay to provide fault tolerance between virtual machines. In contrast to systems such as Hypervisor and ReVirt [11] which enforce determinism at the ISA level, Slingshot enforces determinism at the application level. This choice was driven by our desire to use a robust commercial virtual machine (VMware) without modification. Our approach to enforcing determinism was inspired by Rodrigues' BASE [24], which provides Byzantine fault tolerance by wrapping non-deterministic software with a layer that enforces deterministic behavior. A similar approach was used in Brown's operator undo [7].

Slingshot is an instance of cyber foraging [1], the oppor-

Service	Remote Execution	Slingshot				
		Steady-State	Creating 1st Replica		Creating 2nd Replica	
			Warm Cache	Cold Cache	Warm Cache	Cold Cache
Speech w/o microdrive	0.67 (0.67–0.67)	0.24 (0.24–0.24)	0.69 (0.68–0.70)	0.69 (0.67–0.73)	0.52 (0.51–0.52)	0.52 (0.51–0.52)
Speech with microdrive	0.67 (0.67–0.67)	0.24 (0.24–0.24)	0.80 (0.79–0.81)	0.80 (0.79–0.81)	0.65 (0.65–0.65)	0.65 (0.63–0.68)
VNC w/o microdrive	18.9 (18.9–19.0)	7.4 (7.2–7.5)	22.8 (21.5–23.6)	22.2 (19.9–23.6)	9.8 (9.7–9.9)	10.0 (9.9–10.0)
VNC with microdrive	18.9 (18.9–19.0)	7.4 (7.2–7.5)	24.1 (23.9–24.3)	23.1 (21.6–24.4)	13.8 (13.0–14.4)	14.6 (14.4–14.8)

This figure summarizes the average response time (in seconds) for all experiments. Each entry shows the mean of three trials, with the low and high trials given in parentheses. The second column shows response time for remote execution on the home server. The third column shows steady-state performance for Slingshot with a replica on the nearby surrogate. The remaining columns show response time while instantiating a replica on the nearby surrogate with and without a replica running on the distant surrogate.

Figure 12. Summary of response time results

Service	Creating 1st Replica		Creating 2nd Replica	
	Warm Cache	Cold Cache	Warm Cache	Cold Cache
Speech w/o microdrive	28:06 (27:50–28:27)	27:55 (27:50–28:04)	28:10 (28:05–28:11)	27:57 (27:57–27:58)
Speech with microdrive	3:35 (3:32–3:40)	3:27 (3:26–3:28)	3:39 (3:34–3:45)	3:33 (3:32–3:34)
VNC w/o microdrive	27:48 (27:07–28:28)	27:58 (27:12–28:45)	31:16 (30:57–31:31)	31:08 (31:00–31:25)
VNC with microdrive	6:37 (6:20–7:13)	7:29 (6:59–8:27)	8:59 (8:01–10:00)	8:20 (6:47–9:00)

This figure summarizes the time (in minutes) to create a new replica on the nearby surrogate for all experiments. Each entry shows the mean of three trials, with the low and high trials given in parentheses. The second and third columns show the time to instantiate a replica when no replica runs on the distant surrogate. The last two column show results with a replica on the distant surrogate.

Figure 13. Summary of replication time results

tunistic use of surrogates to augment the capabilities of mobile computers. Previous work in Spectra [12] examined how a cyber foraging system could locate the best server and application partitioning to use given dynamic resource constraints. In contrast, Slingshot takes this selection as a given and provides a mechanism for utilizing surrogate resources. More recently, Balan [2] and Goyal [16] have also proposed cyber foraging infrastructure. Compared to these systems, the major capability added by Slingshot is the ability to execute stateful services on surrogate computers. Data staging [13] and fluid replication [19] use surrogates to improve the performance of distributed file systems. They share common goals with Slingshot such as minimization of latency and ease of management—however, Slingshot applies these principles to computation rather than storage.

The applications we have investigated so far have been easy to partition because they were designed for client-server computing. Potentially, Slingshot could use one of several methods that automatically partition applications. For instance, Coign [18] partitions DCOM applications into client and server components. Globus [14], Condor [5], and Legion [17] dynamically place functionality, but target grid rather than mobile computing.

7 Conclusions and future work

Handhelds can improve interactive response time by leveraging surrogate computers located at wireless hotspots. Slingshot's use of replication offers several improvements over a strategy that simply migrates remote services between computers. Replication provides good response time for mobile users who move between wireless hotspots; while a new replica is being instantiated, other replicas continue to service user requests. Replication also lets Slingshot recover gracefully from surrogate failure, even when running stateful services.

Slingshot minimizes the cost of operating surrogates. For these computers to be of maximum benefit, they must be located at wireless hotspots, rather than in machine rooms that are under the supervision of trained operators. Slingshot uses off-the-shelf virtual machine software to eliminate the need to install custom operating systems, libraries, or applications to service mobile users. All application-specific state associated with each service is encapsulated within its virtual machine. Further, Slingshot's replication strategy means that surrogates need not provide 24/7 availability. If a surrogate fails or is rebooted, no state is lost.

Harnessing surrogate computation is a complex problem. Slingshot currently provides several pieces of the puzzle, including the use of replication to improving response time and the elimination of hard surrogate state to improve ease of management. Other pieces of the puzzle remain. Slingshot does not yet address the privacy issues inherent to running computation on third-party hardware. Trusted computing efforts [15] provide promise in this area. Slingshot does not provide a mechanism for securely controlling replica instantiation and termination. Other areas of potential investigation are load management and policies for creating and destroying replicas. We believe that Slingshot will be an extremely useful platform on which to conduct such investigations.

Acknowledgments

We are grateful to Mike Kozuch and the ISR team for providing us with the Fauxide and Vulpes components used in this work. We also thank Manish Anand, Edmund B. Nightingale, Daniel Peek, the anonymous reviewers, and our shepherd, Doug Terry, for comments that helped improve this paper. This research was supported by CAREER grant CNS-0346686 from the National Science Foundation and by an equipment grant from Intel Corporation. The views and conclusions in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of NSF, Intel, the University of Michigan, or the U.S. government.

References

[1] BALAN, R., FLINN, J., SATYANARAYANAN, M., SINNAMOHIDEEN, S., AND YANG, H.-I. The case for cyber foraging. In *the 10th ACM SIGOPS European Workshop* (Saint-Emilion, France, September 2002).

[2] BALAN, R. K., SATYANARAYANAN, M., PARK, S., AND OKOSHI, T. Tactics-based remote execution for mobile computing. In *Proceedings of the 1st Annual Conference on Mobile Computing Systems, Applications and Services* (San Francisco, CA, May 2003), pp. 273–286.

[3] BARATTO, R. A., POTTER, S., SU, G., AND NIEH, J. MobiDesk: Mobile virtual desktop computing. In *Proceedings of the 10th Annual Conference on Mobile Computing and Networking* (Philadelphia, PA, Sept/Oct 2004), pp. 1–16.

[4] BARHAM, P., DRAGOVIC, B., FRASER, K., HAND, S., AND HARRIS, T. Xen and the art of virtualization. In *Procedings of the 19th ACM Symp. on Operating Systems Principles* (Bolton Landing, NY, October 2003), pp. 164–177.

[5] BASNEY, J., AND LIVNY, M. Improving goodput by coscheduling CPU and network capacity. *International Journal of High Performance Computing Applications 13*, 3 (Fall 1999).

[6] BRESSOUD, T. C., AND SCHNEIDER, F. B. Hypervisor-based fault-tolerance. In *Proceedings of the 15th ACM Symposium on Operating System Principles (SOSP)* (Copper Mountain, CO, December 1995), pp. 1–11.

[7] BROWN, A. B., AND PATTERSON, D. A. Rewind, repair, replay: Three R's to dependability. In *the 10th ACM SIGOPS European Workshop* (St. Emilion, France, September 2002).

[8] CARSON, M. *Adaptation and Protocol Testing thorugh Network Emulation.* NIST, http://snad.ncsl.nist.gov/itg/nistnet/slides/index.htm.

[9] CHEN, P., AND NOBLE, B. When Virtual is Better Than Real. In *Proceedings of the 8th IEEE Workshop on Hot Topics in Operating Systems* (Schloss Elmau, Germany, May 2001).

[10] COX, L. P., MURRAY, C. D., AND NOBLE, B. D. Pastiche: Making backup cheap and easy. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation* (December 2002), pp. 285–298.

[11] DUNLAP, G. W., KING, S. T., CINAR, S., BASRAI, M. A., AND CHEN, P. M. ReVirt: Enabling intrusion analysis through virtual-machine logging and replay. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation* (December 2002), pp. 211–224.

[12] FLINN, J., NARAYANAN, D., AND SATYANARAYANAN, M. Self-tuned remote execution for pervasive computing. In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII)* (Schloss Elmau, Germany, May 2001), pp. 61–66.

[13] FLINN, J., SINNAMOHIDEEN, S., TOLIA, N., AND SATYANARAYANAN, M. Data staging for untrusted surrogates. In *Proceedings of the 2nd USENIX Conference on File and Storage Technology* (San Francisco, CA, March/April 2003), pp. 15–28.

[14] FOSTER, I., KESSELMAN, C., NICK, J., AND TUECKE, S. Grid services for distributed system integration. *Computer 35*, 6 (2002).

[15] GARFINKEL, T., PFAFF, B., CHOW, J., ROSENBLUM, M., AND BONEH, D. Terra: A virtual machine-based platform for trusted computing. In *Procedings of the 19th ACM Symp. on Operating Systems Principles* (Bolton Landing, NY, October 2003), pp. 193–206.

[16] GOYAL, S., AND CARTER, J. A lightweight secure cyber foraging infrastructure for resource-constrained devices. In *Proceedings of the 6th IEEE Workshop on Mobile Computing Systems and Applications* (Lake Windermere, England, December 2004).

[17] GRIMSHAW, A. S., AND WULF, W. A. Legion: Flexible support for wide-area computing. In *Proceedings of the 7th ACM SIGOPS European Workshop* (1996).

[18] HUNT, G. C., AND SCOTT, M. L. The Coign automatic distributed partitioning system. In *Proceedings of the 3rd Symposium on Operating System Design and Implemetation (OSDI)* (New Orleans, LA, February 1999), pp. 187–200.

[19] KIM, M., COX, L. P., AND NOBLE, B. D. Safety, visibility, and performance in a wide-area file system. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies* (January 2002).

[20] KOZUCH, M., AND SATYANARAYANAN, M. Internet Suspend/Resume. In *Proceedings of the 4th IEEE Workshop on Mobile Computing Systems and Applications* (Callicoon, NY, June 2002).

[21] LI, J., KROHN, M., MAZIRES, D., AND SHASHA, D. Secure untrusted data repository (SUNDR). In *Proc. of the 6th Symp. on Op. Syst. Des. and Imp.* (San Francisco, CA, December 2004), pp. 121–136.

[22] MICROSOFT CORPORATION. *Universal Plug and Play Forum*, June 1999. http://www.upnp.org.

[23] OSMAN, S., SUBHRAVETI, D., SU, G., AND NIEH, J. The design and implementation of Zap: A system for migrating computing environments. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation* (December 2002), pp. 361–376.

[24] RODRIGUES, R., CASTRO, M., AND LISKOV, B. BASE: Using abstraction to improve fault tolerance. In *Proceedings of the 18th Symposium on Operating Systems Principles (SOSP)* (Banff, Canada, October 2001), pp. 15–28.

[25] SAPUNTZAKIS, C. P., CHANDRA, R., PFAFF, B., CHOW, J., LAM, M. S., AND ROSENBLUM, M. Optimizing the migration of virtual computers. In *Proceedings of the 5th Symposium on Operating System Design and Implementation* (Boston, MA, December 2002), pp. 377–390.

[26] TOLIA, N., HARKES, J., KOZUCH, M., AND SATYANARAYANAN, M. Integrating portable and distributed storage. In *Proceedings of the 3rd Annual USENIX Conference on File and Storage Technologies* (San Francisco, CA, March/April 2004).

[27] TOLIA, N., KOZUCH, M., SATYANARAYANAN, M., KARP, B., BRESSOUD, T., AND PERRIG, A. Opportunistic use of content addressable storage for distributed file systems. In *Proceedings of the 2003 USENIX Annual Technical Conference* (May 2003), pp. 127–140.

[28] WALDSPURGER, C. A. Memory resource management in VMware ESX server. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation* (Boston, MA, December 2002), pp. 181–194.

[29] WHITAKER, A., SHAW, M., AND GRIBBLE, S. D. Scale and performance in the Denali isolation kernel. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation* (Boston, MA, December 2002), pp. 195–209.

DeltaCast: Efficient File Reconciliation in Wireless Broadcast Systems

Julian Chesterfield
University of Cambridge, Computer Laboratory, Cambridge, UK
julian.chesterfield@cl.cam.ac.uk

Pablo Rodriguez
Microsoft Research, 7 JJ Thompson Avenue, Cambridge, UK
pablo@microsoft.com

Abstract

Recently, there has been an increasing interest in wireless broadcast systems as a means to enable scalable content delivery to large numbers of mobile users. However, gracefully providing efficient reconciliation of different versions of a file over such broadcast channels still remains a challenge. Such systems often lack a feedback channel and consequently updates cannot be easily tailored to a specific user. Moreover, given the potentially large number of possible versions of a file, it is impractical to send a tailored update for each particular user.

In this paper we consider the problem of efficiently updating files in such wireless broadcast channels. To this extent, we present DeltaCast, a system that combines hierarchical hashes and erasure codes to minimise the amount of battery power and the amount of time needed to synchronise each mobile device. Based on our experimental results, we show that DeltaCast is able to efficiently identify the missing portions of a file and quickly updated each client.

1 Introduction

Wireless communication systems are one of the fastest growing areas of communication technology, with new devices and standards constantly emerging, developed to transmit digital signals over both short and long distances. There are two main delivery approaches for wireless data services: *point-to-point* and *point-to-multipoint* or broadcast systems [1]. Point-to-point systems employ a basic client-server model, where the server is responsible for processing a query and returning the result to the user via a dedicated point-to-point channel. In broadcasting or *DataCasting* systems, on the other hand, the server actively pushes data to the users in an open loop fashion. The server determines the data to be transmitted and schedules the broadcast. A user listens to the broadcast channel to retrieve data without any signalling to the server and thus is responsible for his own query processing.

Point-to-point access is particularly suitable for light loaded systems where contention for wireless resources and server processing is not severe. As the number of users increases, however the overall system performance can quickly degrade. Compared with point-to-point access, broadcasting is a very attractive alternative [2][3]. It allows simultaneous access by an arbitrary number of mobile clients without causing any inter-receiver interference and thus allows efficient usage of the scarce wireless bandwidth and server resources.

Wireless data broadcast services have been available as commercial products for many years (e.g., StarBand [4] and Hughes Network [5]). However, recently there has been a particular push for Wireless DataCasting systems both in Europe and in the US. Both 3gpp as well as 3gpp2 are developing plans to provide multicast/broadcast support over UMTS/CDMA networks [6] [7]. Moreover, systems such as Digital Audio Broadcast (DAB) and Digital Video Broadcast (DVB) standards in Europe [8] and ATSC in the US which have been traditionally used for the transmission of digital radio and television, are also capable of providing data services, a feature that is anticipated to experience significant utilisation. One of the key motivators for the rapid growth of digital broadcasting and in particular, data services over DAB and DVB, is that cell phone manufacturers are announcing plans to trial new devices incorporating built-in DAB/DVB receivers over the next year [9]. This will enable a wide range of DataCasting services into mobile devices. Another example of DataCasting services is the smart personal objects technology (SPOT) by Microsoft [10], which further highlights the industrial interest in and feasibility of utilising broadcast for wireless data services. Using such DataCasting services, mobile devices can continuously receive timely information such as maps, software, news, weather, traffic information, etc.

In this paper, we focus on wireless data broadcasting services and mechanisms for efficiently reconciling content on distributed receiving devices to the latest available version. Reconciliation of data files between two computers is a well studied problem with efficient solutions such as rsync [11] utilised on a wide scale in the Internet. Such solutions, however, rely on two-way communication channels to identify the appropriate delta information between two files and are best suited for point-to-point reconciliation. In DataCasting systems, mobile users with intermittent coverage or settings may synchronise at arbitrary times, thus, generating a potentially very large number of different versions. DataCasting systems often lack a feedback channel, which prevents the server from gathering information about the user's file and therefore provide a tailored delta update. Even if the server had knowledge of which versions each client has, it is difficult to tailor the broadcast channel to a specific user without delaying other users with different versions. All these conditions make it hard to construct a system that gracefully provides differential updates to a large number of different users in a multicast/broadcast system.

One possible solution is to make sure that the users update their own data to the most recent version every time by simply transmitting the latest version of the file. It is generally expected, however, that changes to content versions of a file on average are likely to be incrementally small, hence the motivation for widespread adoption of protocols like rsync, providing a good deal of redundancy between current and previous data. By forcing the user to repeatedly download the same portions of the file, inefficient battery consumption and an increase in download latency can be experienced as a result.

Up until recently, fixed power devices have been the most common DataCasting service receivers, e.g. home radios and computers, or vehicular based receivers. In the near future however it is anticipated that there will be a proliferation of small, handheld, mobile receivers such as cellphones and PDAs capable of receiving DataCasting services. Access efficiency and energy conservation therefore are two critical issues for users in a wireless data broadcast system. Access efficiency refers to maximising the speed with which data is downloaded, while energy conservation minimises the mobile client's power consumption when accessing the data of interest.

In this paper we present *DeltaCast*, a new mechanism that provides efficient file reconciliation in DataCasting systems. DeltaCast combines decomposable hierarchical hashing schemes with low-cost random linear codes to enable efficient file synchronisation among an unlimited number of receivers. As we will see later in the paper, DeltaCast has the ability to update different receivers' versions simultaneously through the use of *master* encoded deltas that can be used by any receiver at any point in time.

The rest of the paper is structured as follows. Section 2 discusses the design of *DeltaCast* in response to the specific challenges of the environment. We discuss the use of decomposable hashes and erasure codes to minimise the amount of data downloaded by each receiver. Section 3 we outline the performance considerations of the system, and our approach toward measuring the benefits and trade-offs of different file synchronisation techniques. We present results from our experiments with the *DeltaCast* system, comparing the optimised *DeltaCast* against alternative approaches, and demonstrating that it is a very efficient approach to reconciling data in broadcast systems. In section 4 we discuss related work, and we conclude the paper in section 5.

2 DeltaCast Design

DeltaCast is an efficient file synchronisation protocol for one-way, receiver driven incremental file updating. A typical use of DeltaCast would be in the radio broadcast environment, where one-to-many DataCasting can provide a highly effective means to reach unlimited numbers of receivers without requiring any increase in network capacity to scale the service to larger numbers. *Deltacast* in fact is not just useful within the wireless broadcast environment, we expect it to provide similar benefits in terrestrial point-to-multipoint environments such as IP Multicast, Content Distribution Networks and Peer-to-peer overlay systems.

A limitation of such radio broadcast systems, however is that data can only be delivered asymmetrically. Typically, receiving devices do not have any return channel to the source, and therefore the transmission must be uni-directional. Typical applications in this environment might be providing downloads of local information to mobile devices, maps, software, video, etc. An advantage of DataCasting is that multiple channels can be used to transmit content in parallel, and the scalability of the system is only constrained by the amount of content that can fit in the available spectrum bandwidth. Data content is typically carouseled continuously to enable receivers to 'tune-in' at any stage to the relevant channel and receive content. Since this is a broadcast only environment, low layer errors are handled through strong redundant Forward Error Correction (FEC) applied to the source signal. In this paper we will assume that the receiver's application does not experience errors due to signal interference and that any missing data is due to lack of application level synchronisation.

Due to the heterogeneity of the receiver group and the wireless environment, it is likely that a) there will be intermittent connectivity for individual receivers (e.g. loss

of signal, power-down overnight etc.) and b) that different receivers start the synchronisation process at different points in time, resulting in a high variation in most recently synchronised versions of data distributed across the whole receiver set. This variability is compounded by the fact that the source has no knowledge of the exact distribution of data versions. In addition to which, many devices utilising such a service are likely to be small, mobile and consequently power constrained. The solution must take all these considerations into account.

2.1 Problem Definition

The setup for the file synchronisation problem is as follows. We have a current file f_{new} with size F, and a set of N outdated file versions $f_{old,j}$, for $j \in 1,..,N$ over some alphabet $\sum$, and two types of machines C (the clients) and S (the server) connected through a broadcast data channel and no feedback link. We assume that machines in C have copies in $f_{old,j}$ and S only has a copy of f_{new}, and the goal is to design a protocol that results in C holding a copy of f_{new}, while minimising the amount of time and the resources consumed at the clients.

2.2 General System Overview

The Deltacast system comprises two distinct phases; the *detection* of changes between old and new content and the *update* of such changes. During the detection phase, receivers determine what portions of the local file match the target updated version at the server and can be re-utilised. Note that the receiver may not match any portion of the local file, in which case it requires to download the full file. Unmatched portions are discarded, while matched portions are kept in a temporary file. During the update phase, each receiver downloads enough data to fill in the gaps left by the un-matched portions.

The server receives no feedback at all from the receiver group and sends out a constant stream of hashes and data. Hashes are used during the detection phase, while data is used during the update phase. During the detection and the update phase, erasure encoding is used. Such erasure codes ensure that any information received by any receiver is useful.

To identify the unmatched portions with a high level of granularity, the server does a multi-level hierarchical division of the target file into blocks of a fixed size. The size of a block at each hierarchy level being half the size of the preceding level, i.e. decreasing powers of 2. For every block of data the server then generates a small, unique hash. When all the hashes at a given level have been generated, the server uses such hashes to create *erasure hashes* for a particular level. There is a potentially very large number of unique erasure hashes that can be

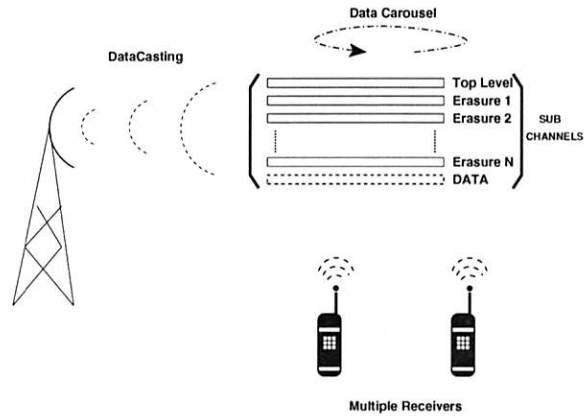


Figure 1: DeltaCast Architecture.

produced, which are published over the broadcast channel (normally erasure hashes for each level are sent over different channels). The same process is then repeated at each level of the hierarchy, producing a constant stream of erasure hashes at each level of the hierarchy. At the bottom level of the hierarchy, the server also produces *encoded data blocks*, which are broadcast in a separate channel and are used by the receivers to fill in the unmatched portions of their local files. Figure 1 shows the overall architecture of the DeltaCast system.

Receivers in the detection phase download enough hashes at each hierarchy level to determine the unmatched portions of the local file. To this extent, receivers first tune in to the top-level hash channel and download a complete set of hashes. Typically the top-level hashes are sent un-encoded since no additional benefit is achieved by encoding them. The receivers know the hashing algorithm used by the server, and thus, can generate a local database of hierarchical hashes corresponding to the local file. By comparing each downloaded hash against the local database, the receiver identifies the high level areas of the file that are matched. For each area of the file where the hash comparison failed, the receiver downloads enough erasure hashes from the subsequent level (normally, the number of erasure hashes required is twice the number of un-matched hashes at the previous level). Such erasure hashes are then combined with the hashes of the matched blocks at that particular level in order to reconstruct the missing hashes for the un-matched blocks. Since the hashes at each subsequent level correspond to a smaller area of the content, the detection of changes becomes more fine-grained each time.

Once the receiver has either reached the lowest level of hashes, or has identified that there is no further benefit to detecting finer-grained changes by reconstructing a

further sub-level of hashes, it enters the *update* phase. In the update phase, for every unmatched hash at the lowest level, the receiver downloads a corresponding number of erasure data blocks. Given the erasure blocks, and the set of matched content blocks, the receiver can then reconstruct the latest version of the content as published by the server. The final file is verified with a strong signature provided by the server.

2.3 The Details

DeltaCast enables receivers to efficiently identify changes in their own content and download only as much data as required to synchronise their version with the source. The most common techniques for providing identifying matched portions between a source and receiver involve content hashing over partial portions of a file. Due to the high probability of data similarity in incremental version updates, forwarding block hashes, can potentially save a significant amount of bandwidth and time through identifying the precise portions of the old version which need updating, rather than forwarding the whole data file. However, identifying the most efficient block size for which the source sends hashes to the receiver, is challenging and greatly varies depending upon the nature of the content changes. Utilising small block sizes increases the number of blocks and consequently the amount of hashes transmitted. In contrast, utilising large block sizes decreases the amount of hash data transmitted, but lowers the precision of each hash, consequently reducing the ability to detect fine-grained data changes. In this manner there is a distinct trade-off between the granularity of the hashes and the amount of data transmitted, the optimal setting of which can only be determined retrospectively by comparing the two versions.

One way to solve this problem is to use a *hierarchical hashing* scheme that gradually decreases the block sizes to narrow down the unmatched portions. However, hierarchical hashing schemes often rely on multiple rounds of communication between the server and the client, sending only the appropriate hashes at each hierarchy level. Such multi-round protocols cannot be implemented in open-loop DataCasting systems. Moreover, since the server does not know which versions are currently held by the receivers, it needs to broadcast all hashes for each block at every hierarchy level. This wastes precious bandwidth resources at the broadcast channel but, even more significantly, increases the average time for a user to update its file since many hashes in the carousel will correspond to already matched file portions.

The DeltaCast approach is to utilise a hierarchical hashing scheme combined with highly efficient *erasure coding* and *decomposable hashes* in order to efficiently reconcile different file versions at multiple receivers simultaneously.

The approach is as follows:

1. The source performs a hierarchical division of the file into n different layers $i \in 1,..,n$, with each layer having a different block size $b_{max}, ..., b_{min}$, where b_{max} is the block size at the top layer $i = 1$, and b_{min} is the block size at the lowest layer $i = n$. In this paper we assume that the block size is halved at each level, i.e. $b_{i-1} = b_i * 2$. For each block at a given level, the server calculates a block hash $h(b)$, which is of size H bits. The only constraint on block sizes is that $b_{min} > H$.

2. For the first level of the hierarchy, the server transmits the top-level hashes continuously. For each subsequent level of the hierarchy, the server calculates a large number of *erasure hashes* based on the block hashes at that level. For a given level, erasure hashes are calculated as a random linear combination of the hashes at that particular level and have the property that they can be used in place of any hash. The total number of possible different erasure hashes is very large. Erasure hashes for each level are continuously transmitted in a separate channel.

3. One extra channel is included to forward the data content. Data content is not sent in its original form but as erasure blocks of the original data.

The reconstruction at the receiver proceeds as follows:

1. Let F be the size of the file at the server. The receiver first downloads F/b_{max} top level hashes and coarsely identifies the matching blocks. To do this, it checks the values across the whole file using a rolling block window and marks any unchanged data blocks. The set of blocks where the hashes do not coincide are marked as unmatched. The size of that set at level i is represented by $u(i)$.

2. For each level $i > 1$ of hash values, the receiver downloads $u(i-1) * 2$ erasure hashes in order to reconstruct the correct set of hashes at level i. As we will see later in this section, we can halve the amount of erasure hashes downloaded at level i using decomposable homomorphic hashes. Such hashing functions hold the property that $h(a) = h(b) + h(c)$ where a is the top level block, and b and c are the sub-level blocks corresponding to the top-level block a. This property enables us to reconstruct all hashes at level i by downloading only $u(i-1)$ erasure hashes.

3. The same process is repeated at each level of the hierarchy. After decoding the hashes at level n and identifying the number of failing blocks $u(n)$, the receiver downloads $u(n)$ encoded data blocks from the data content channel. After proper decoding, the receiver is able to reconstruct the file and thus synchronise it's local version with the source file.

Figure 2 illustrates the encoding approach at the source, highlighting the hash relationship between levels, and the decreasing size of the blocks.

2.4 Hash Accuracy

One important parameter in the DeltaCast system is determining the size of the hash H. Provided that the probability distribution of hash values is perfectly random, the statistical accuracy of a hash of H bits can be expressed as a function of the comparison file at the receiver, f_{old}, of length m and the block size b on which the hash is based. Suppose that S initially sends a set of H-bit hashes to C, and that C uses these hashes to check for matches in f_{old}. As the comparison block window at the receiver is rolled across every byte position in the file, there are $m - b + 1$ possible match positions in f_{old} that need to be checked, and for each position there is a certain probability of a false match. Other hashes based on fingerprints [26] could also be used to reduce the complexity of matching blocks over the file, however since these hashes use variable size blocks, trying to maintain a clean hierarchical structure becomes more complex and additional signalling would be required from the source. Since the probability of a certain match occurring at any single position in the file is expressed as $1/2^H$, the probability of any particular match *not* occurring (i.e. no collision) anywhere in the file is $(1 - 1/2^H)^{m-b+1}$. Thus we can say that the chance of a false match is $1 - (1 - 1/2^H)^{m-b+1}$. For an even chance of not getting a false match, the required hash size can be approximated by $lg(m) + log(m/b)$, while $lg(m) + lg(m/b) + d$ bits are needed to get a probability less than $1/2^d$ of having a false match between files.

For every file downloaded, the server also sends a 16-byte MD5 hash of the entire content that allows the client to check the correctness of the final version. Thus, in the unlikely event that collisions occur due to weak block-level hashes, the client can then download the content again.

2.5 Erasure Hashes

To be able to quickly identify the missing data portions, receivers need to download the correct set of block hashes as fast as possible. Different receivers may be missing different data blocks, however and therefore need a different set of hashes. Thus, a common approach is for a server to continuously send the full set of hashes at each level to ensure that all receivers can synchronise their file regardless of what version they may have. If the server knew which receivers where missing what portions, it could devise an more efficient protocol where certain hashes were to be transmitted more frequently than others. However, without any feedback channel, the server has no choice but to send all hashes. As a result, clients must wait for the specific hash to arrive in the carousel, potentially wasting a lot of time.

DeltaCast solves this problem by encoding the original hashes to produce erasure hashes that can be used by *any* receiver in place of *any* missing hash at a given level (a similar thing happens for the data content channel). To this extent, we have implemented a rateless erasure encoder/decoder that generates a set of random linear erasure blocks. Each erasure block is produced by selecting a unique set of co-efficients, applying the co-efficients across the original source blocks and then calculating the sum of all the blocks over a finite field(e.g. GF 2^{16}). As long as the co-efficients are generated randomly over a sufficiently large field, the probability of generating a non-innovative code becomes insignificantly small.

Such codes hold the property that for a system of n source blocks, the receiver only needs to obtain any combination of n erasure or original data blocks, where the blocks are linearly independent, in order to reconstruct the original source data. For the receiver to be able to decode the erasure blocks, the encoded data is also tagged with the set of random co-efficients that were used to generate the linear combination. The co-efficients can be explicitly broadcast with the encoded block, although a more efficient approach would be to use a seeded pseudo-random generator, in which case only the seed value and the index would be required. To improve the decoding time, one can also use well-known sparse encoding techniques which require downloading some extra data but significantly reduce the overall decoding time [36].

2.6 Decomposable Hashes

Decomposable hashes permit receivers to download half the number of erasure hashes at each level. The idea of using decomposable hashes is that if we have already transmitted a hash value for a parent block at a given hierarchy level, then for certain types of hash function we could compute a child hash at the next hierarchy level from the top-level hash and one extra child hash. In practise though, designing appropriate hash functions to implement this is nontrivial.

Assume that a top-level block is created out of bytes $[l, r]$, and the corresponding children blocks are $[l, m]$ and $[m, r]$. We say that a hash function is decompos-

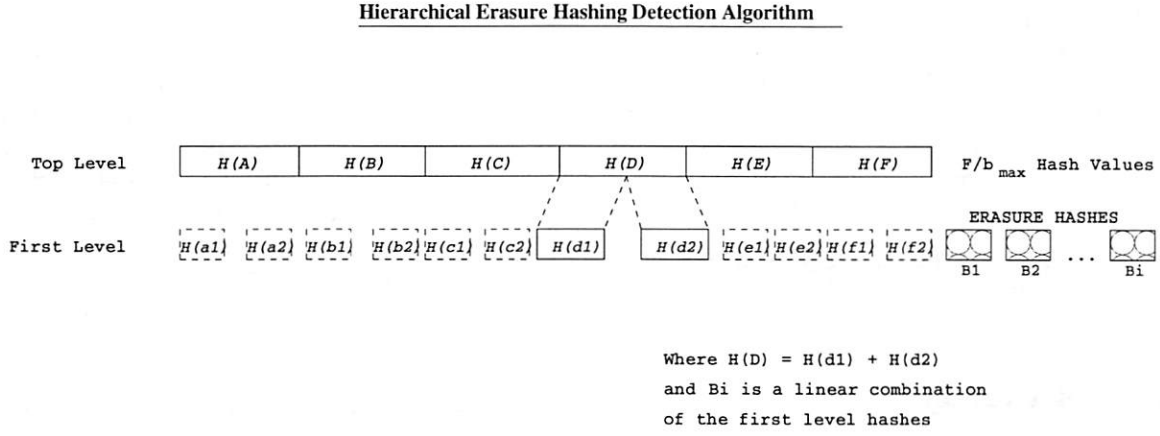


Figure 2: DeltaCast hashing scheme. Two levels.

able if we can efficiently compute $h(f[m+1,r])$ from the values $h(f[l,r])$, $h(f[l,m])$, $r-l$, and $r-m-1$, and also $h(f[l,m])$ from $h(f[l,r])$, $h(f[m+1,r])$, $r-l$, and $m-l$.

A decomposable hash function allows us to significantly save on the cost of delivering hashes for identifying matching data. Since receivers already have a hash for the parent block, they only need to receive one additional hash per pair of sibling blocks, the hash for the other sibling can then be computed from these two. Utilising decomposable hashes is particularly efficient in the deltacast system since erasure hashes can be produced in the same way out of decomposable hashes. In fact, decomposable hashes can be used as input to the erasure decoding algorithm at the next level of the hierarchy where, given the parent hash and an additional erasure block, both of the next level hashes can be reconstructed. Decomposable parent hashes can be interpreted as simple erasure codes created as a linear combination of the two child block hashes.

Decomposable hashes can be implemented using Homomorphic hashing functions such as the ones described in [12][13][14] and [15] where $h(a+b) = h(a) + h(b)$ or $h(a*b) = h(a)*h(b)$, a and b being two different blocks. To make use of decomposable hashes in DeltaCast we ensure that $h(f[l,r])$ for a block at a given level is equal to $h(f[l,m])+h(f[m,r])$, where $h(f[l,m])$ and $h(f[m,r])$ are the hashes for the corresponding sibling blocks at the next hierarchy level. To this extent, we define $h(b_1,...b_n) = \sum_{v=1}^{v=n} g^v \cdot b_n$, where b_i is an individual block, $(b_1,...,b_n)$ is the parent block made of the concatenation of blocks $1,..,n$, and g is a generator number of a given Galois field. A more detailed study of such functions is described in [15].

3 Experimental Results

Next we consider the performance of the algorithm. We evaluate the improvement in data download requirements through utilising hierarchical hashing versus single-layer hashing, and additionally consider the performance from a temporal perspective, comparing the time to synchronise content using Deltacast against a full content download as well as the non-encoded case.

3.1 Performance Considerations

Within a fixed wire environment such as the Internet, the key optimisation metric for data download performance is minimising the overall download latency from initialisation to completion of the data transfer from source to receiver. Factors which typically contribute to this latency are protocol communication overheads, e.g. for multi-round conversations between senders and receivers, the delay for which can vary considerably depending on the link. In the broadcast radio environment, latency is typically tightly controlled and bounded by the physical propagation speed of the signal and the channel capacity. Since the environment can only support one-way broadcast communication, we do not consider any interactive protocol communication overhead. Instead, we must consider the amount of data that is transmitted in order to synchronise the local receiver data version with that of the server. We consider the average time for which a receiver may have to wait to receive sufficient data, and in addition we must consider the overheads of erasure decoding introduced by the Deltacast system. The latter consideration is particularly important in drawing a fair comparison since although the decoding may be handled

offline, it is still power intensive, and whilst we do not measure precise energy consumption values, we include it as a latency factor in our experiments.

3.1.1 Power Efficiency

Minimising receiver power consumption is a responsibility for both the receiver device and the protocol designer. The process of tuning a radio to a channel and demodulating data being transmitted consumes energy. Attempting to minimise the frequency with which a receiver must demodulate data in order to synchronise itself with the transmission from a source is therefore crucial. By utilising erasure coding, Deltacast ensures that *any* data the receiver may demodulate will be useful. DeltaCast leaves a certain amount of intelligent interpretation up to the receiver by providing it with multiple encoded channels from which to select and download data as required. There is no passive listening required while certain data segments rotate through the carousel since any erasure block can be used as data. There is a trade-off in utilising erasure coding in terms of the decoding requirements and consequently additional power is consumed by the device in the process of decoding erasure blocks, however we maintain that there are many optimisations that can be applied to minimise the impact of the decoding complexity such as using sparse matrices, or dividing the source data into smaller files. The benefits of utilising erasure codes therefore greatly outweigh the processing disadvantages, and we demonstrate that the latency overhead of decoding is much lower than the download latency of unencoded data transmission.

3.1.2 Broadcast Channel Capacity

The Datacasting environment is bandwidth constrained due to the finite limitation of regulated wireless spectrum compared to the diversity of content that users might like to see. However, there is provision for data transmission, and there a large variety of applications that are anticipated for use within the environment. Channels are typically assigned in multiples of 64Kbit/s due to the tight correlation with traditional audio and video transmission. Wide-band popular applications might receive more than one channel, enabling faster or more frequent data updates, however typically it is assumed that each channel would be utilised by one data carousel. In our experiments therefore we consider the usable throughput of the channel to approximate close to 50kbit/s, leaving an additional 14Kbit/s to account for protocol headers and any additional signalling information that may be required by the source. Within the channel we further subdivide the data into sub-channels. We do not draw a distinction between physical division of the spectrum or logical, e.g. IP level multicast addressing, since this is highly technology dependent, however we assume that channels can be arbitrarily divided into smaller units.

3.1.3 Latency

Minimising data transmission bandwidth through efficient encoding is highly desirable, however it is also important to consider the delays imposed by the system on a user device. Certain types of information have more real-time constraints than others, e.g. stock quotes, or news sites, and we therefore also address the performance of the system in terms of data synchronisation latency. DeltaCast performs extremely favourably in this respect since the utilisation of erasure codes does not impose any download delays from the data carousel to the receiver, and the actual data that is required is in fact minimal in comparison to any other schemes we have considered. In the following section therefore we also address the latency performance of the system, and present some quantifiable results to outline the trade-off in decoding latency versus channel download latency.

3.2 Experimental Setup

For analysing the performance of Deltacast, we utilised a random subset of a collection of ten thousand web pages available at [16]. The collection comprises multiple text/html pages which were crawled repeatedly every night for several weeks during Fall 2001. Pages were selected at random from two massive web crawls of hundreds of millions of pages and are thus a fairly reasonable random sample of the web. We consider such a data set to be representative of a content subscription based channel such as a news, sports or weather information stream which would likely be composed of textual information, images and meta-data such as HTML formatting tags.

For our experiments we divided collections of pages into *content channels*, each one comprising one hundred sites. In a commercial DataCasting environment, such channels might be tailored to specific types of information, e.g. sports, news, etc. We assumed that distributed across the user group is a wide range of versions of the file, some recent, and some non-existent. Whenever a user wants to sync its version to the source, it updates all the pages within a *content channel*. For each *content channel* we have four different instances in time: a base set and three updated versions crawled 2, 20, and 72 days later, respectively. Our experiments always considered the cost of updating an earlier version to one of the more recent versions.

We also measured the impact of different types of content such as source code distributions and binary files. We consider that these content types are representative of

alternative applications such as geographical map information and software updates for mobile devices. We ran experiments for multiple content channels and averaged the performance over all content channels, the results of which are presented in this section.

To evaluate the performance of DeltaCast we will first study what is the impact of the number of hash levels on the overall performance of the system for different types of file. Too few hash levels may not provide enough granularity while too many hash levels may waste precious bandwidth to download unnecessarily detailed hashes. Next, we compare the performance of an optimised DeltaCast system with that of a flat hash system that only transmits one level of hashes. We then study the impact of using DeltaCast to concurrently update users with different file versions. Finally, we analyse the latency benefits of using DeltaCast to update users that join the data carousel at random times and determine the overhead of the decoding times imposed by using encoded data.

3.3 Number of Hierarchy Levels

We next evaluate the performance trade-offs of utilising multiple hash layers in DeltaCast. To this extent we compared the performance of DeltaCast for multiple hash levels when reconciling a content channel of 1.7 MBytes of source data incorporating 100 Web pages. We also measured the performance on a binary file of size 1.9 MBytes. The motivation being that as we will see next, the optimal number of hashing hierarchies depends upon the nature of the content.

We consider a DeltaCast system that uses a 6 Byte hash size, and a hash-hierarchy that uses a top-level block size of 1024 Bytes, and a bottom-level block size of 8 Bytes. Using such hash sizes we guarantee a probability of collision smaller than 2^{20}. (we did not observe any collisions on our data set due to our use of 7-byte hashes). Each level of the hierarchy has a block size that is half the size of the previous level. For each hash-level we use decomposable hashes. We note that although utilising blocks as small as 8 bytes on html content does provide improvements in bandwidth efficiency, the decoding overhead for such block sizes is very large making the use of such small block sizes infeasible. We therefore focus our attention more on 16 Byte blocks and higher.

Figure 3 illustrates the impact of the number of hierarchy levels on the amount of data required to reconcile a set of Web pages. For each combination of hierarchy levels we have illustrated the division of hash data and content data out of the total data downloaded. We can clearly see that in utilising a smaller number of hierarchy levels, the amount of data required is quite high since hashes are not able to efficiently identify file

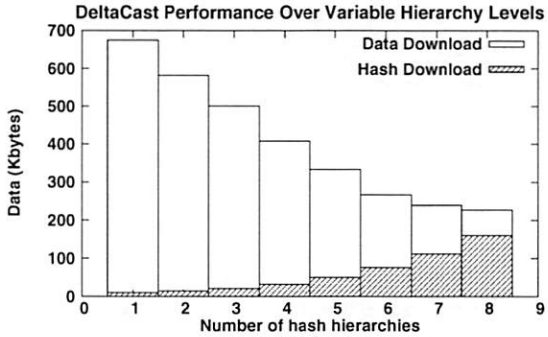


Figure 3: DeltaCast for different Hash Hierarchy Levels - Web content. Both the new and the outdated content approximates to 1.7 MBytes worth of data each. This figure demonstrates that utilising finer-grained hashing (i.e. a greater number of hierarchy levels) is particularly efficient in this context.

changes. As the number of hierarchy levels increases, the amount of hashes downloaded increases, but the overall total data required decreases significantly. The reason for this is that deeper hashes provide a finer level of granularity providing greater clarification of the exact file changes enabling a higher degree of redundancy detection between versions and consequently requiring a smaller amount of actual data downloaded.

From Figure 3 we can see that DeltaCast receivers can use up to eight hierarchy levels and still enjoy a reduction in the total data download. The reason for this is that changes in Web pages are typically on the order of several KBytes, thus, one can use smaller lower block sizes and still find duplicate data portions. Obviously, the number of hierarchy levels cannot go below the point where the block size equals the hash size since the overhead of downloading hashes would be prohibitive. In this example, for an eight-level hierarchy, DeltaCast receivers only need to download about 240 KBytes of data and hashes, which roughly corresponds to 11.4% of the file.

For a file that statistically incorporates changes in larger block sizes such as appending data to the start of a file, or adding a binary module to a software distribution, utilising smaller hash hierarchies will not provide any finer granularity in change detection thereby not reducing the amount of final data blocks downloaded. Applying unnecessary levels of hierarchy in this manner actually increases the bandwidth overhead. To better understand the variable impact of the number of hierarchies based on the content type, figure 4 illustrates the reconciliation of two different versions of a binary file, where the difference in the file lengths was on the order of 135 KBytes. From this Figure we can see that using too many hierarchy levels wastes precious bandwidth since hashes

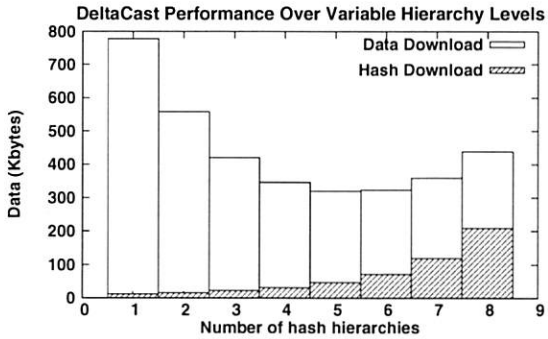


Figure 4: DeltaCast for different Hash Hierarchy Levels - Binary content. Both the new and the outdated content approximates to 2 MBytes worth of data each. This figure demonstrates that in this context (binary content with larger block size changes than HTML data) utilising finer -grained hashing (i.e. a greater number of hierarchy levels) does not provide any additional benefit below an original data block length of 64 Bytes.

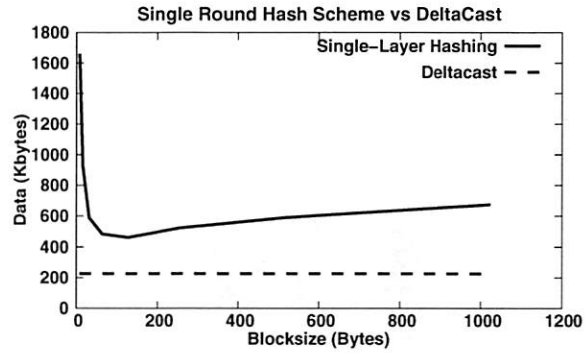


Figure 5: The Single-Layer hashing approach with Different Block Sizes.The figure illustrates the trade-off in hash accuracy versus amount of data downloaded. For comparison, we include the performance of deltacast, however since the algorithm uses hierarchical blocksize hashing in each round we illustrate a constant value for data downloaded.

are unnecessarily downloaded, providing no additional help in identifying missing portions. The optimal level of the hierarchy in this example is around five levels, which is quite different from the case of Web content. Ideally our system should attempt to make some intelligent decision about the optimal number of hashing hierarchies to apply.

To ensure that nodes do not download an unnecessary amount of hashes, DeltaCast servers could use a simple rule that limits the number of hierarchy levels depending on the content type, thus, allowing more levels for content that would be expected to incorporate finer changes, and less levels for content with less fine grained changes and lower intra-file redundancy such as with binary data. Another way to tackle this problem is to allow the DeltaCast receivers to *dynamically* choose the number of hierarchy levels that they want to use and halt the hierarchical detection process when needed. Receivers constantly monitor the new amount of changes detected with each new hash hierarchy. When this value goes below a certain threshold, receivers stop downloading further hash levels and begin downloading the missing data blocks. Such a scheme adapts well to different types of content but also to different types of users with multiple file versions. Each user with a different version will dynamically pick the right hierarchy level that best matches its level of changes. This receiver-driven adaptive approach also provides an efficient way to limit the overhead of deltacast when no redundancy exists between the local and the published content. For instance, in the worst case where no redundancy exists, the receiver would only download 2 levels of hashes in order to identify the lack of usable content and before switching to the simple file download case, which in the example in figure 3 would result in an overhead of just 0.8%.

3.4 Flat Hash System

In this section we compare DeltaCast with a naive hashing scheme that sends only one-level of hashes. For this flat hash system, we vary the block size to determine how an ideal system that could pre-determine the most efficient data block size for all receivers would perform compared to DeltaCast (note that this is an unrealistic scenario but useful for the purposes of a best-case comparison).

Figure 5 shows the total data downloaded to synchronise a collection of Web pages for both a single-layer hashing scheme and DeltaCast. For the single-layer scheme we vary the block size from 8 Bytes to 1024 Bytes. For DeltaCast we use an 7 level hierarchy with top-level blocks equal to 1024 Bytes and bottom-level blocks equal to 16 Bytes. For both schemes we use a hash of size 6 Bytes. From this Figure we can see that the single-layer scheme suffers from large overheads for very small block sizes since the amount of hashes downloaded is very high. Similarly, when the block size is very large, the number of hashes is significantly reduced but missing portions are only coarsely matched, thus resulting in very large portions of redundant data being downloaded. For this specific set of data, the optimal working point for a single-layer scheme is around 128 Bytes.

When comparing the optimal working point of a single-layer scheme with DeltaCast, we see that DeltaCast is much more efficient. The DeltaCast hierarchical system does not need to download deep-level hashes for already matched blocks and is still able to efficiently lo-

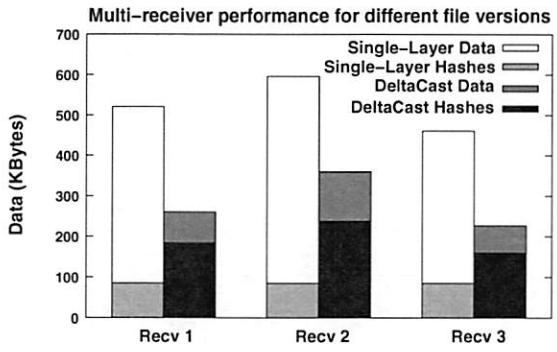


Figure 6: Performance Comparison for Multiple Receivers with Different Content

cate the high-level unmatched data portions. From Figure 5 we see that the performance improvement of DeltaCast compared to the optimal single-layer protocol is greater than a factor of 2.

3.5 Reconciling Different Versions

We now consider the case of three receiver groups with different versions of the same content, all of which are older than the current version being released at the source. We measure the performance of the schemes based on 4 versions of 100 web sites as outlined in section 3.2. Figure 6 illustrates the performance improvement provided by DeltaCast using seven layers compared to a single-round hashing scheme using an optimal block size of 128 Bytes.

All file versions are very similar, within 100 KBytes from largest to smallest. Thus, the overhead of sending hashes in the single-round protocol is very similar across file versions. The actual data downloaded for each single-round protocol experiment, however, ranged from 581KB to 450KB. In contrast, however the DeltaCast experiments on average required approximately half as much data, ranging from 351KB to 221KB. In general we would expect more recent files to have less differences and therefore require less data to be transmitted overall. This is clearly the case for version 3, but not so much for versions 1 and 2.

3.6 Latency Considerations

We next consider the benefits of DeltaCasting in terms of the amount of time that a user needs to reconcile its content. When hashes and data are sent unencoded, on average a user may need to wait for long *idle* periods before he can download a specific hash from the carousel. In the worse case, a user may need to wait for the full carousel to be repeated, thus, significantly increasing the average latency to update a file. With DeltaCast, on the other hand, *any* erasure hash can be used by *any* receiver to decode all the hashes on a given level. The same is true for the lowest level of the hierarchy, which carries the data payload packets. With DeltaCast, increasing the number of hash levels to provide better resolution does not significantly affect the overall latency since receivers do not have to wait long idle periods of time for the right hashes to arrive at each hierarchy level. This is not the case with a hierarchical hashing system that does not use erasure hashes. The higher the number of hierarchy levels, the longer a receiver will have to remain idle since idle times are almost certain to occur at each hierarchy level.

Assume a user is receiving data from a carousel that does not use encoded hashes. If the user needs to download m data units from a given level (a data unit can be either a hash, or a data block), and there are a total of N data units at that particular level, we can calculate the probability that a user receives the required m units by round i or before as $F(i) = \frac{\binom{N-m}{i-m}}{\binom{N}{i}}$. Based on this probability distribution, one can easily calculate the average latency before content is updated.

Next we quantify more precisely the latency benefits of using DeltaCast vs using a hierarchical hashing scheme with unencoded hashes and data. To this extent, we implemented an emulator where each hash hierarchy level is sent in a different carousel with the lowest carousel carrying the data payload. We consider four hierarchical levels. We assume that the total rate of the datacasting channel is 50 Kbit/s (which corresponds to the typical throughput of a DAB channel) and that this channel is evenly partitioned between the data payload and the hashing data. Moreover, the rate corresponding to the hashing data is evenly partitioned among all hash levels. We assume that clients join the carousel at random times.

In Figure 7 we show the average time that it takes for a user to update it's local version of Web content to reconcile with that of the server. We consider four different schemes (a) simple full file download with no hashes, (b) single-layer hashing, (c) hierarchical hashing scheme with no encoded data and (d) DeltaCast. The total latency represented in this Figure includes (i) the time to download the hashes or the erasure hashes, (ii) the time to download the missing data, (iii) the *idle* time waiting in the different carousels for the specific hashes and data to arrive, and (iv) the time to decode the encoded hashes and data. Note that not all latency factors are part of every scheme. For instance, the total latency in scheme (a) is determined by the latency factor (ii). The latency in

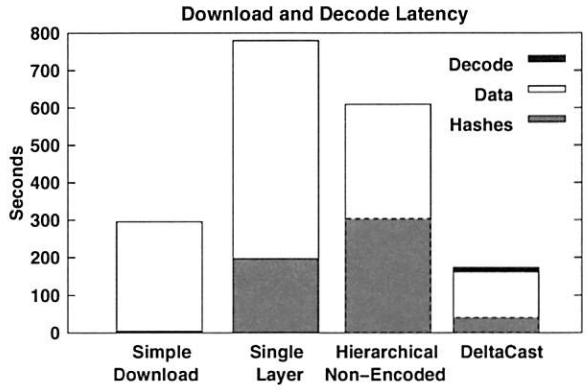


Figure 7: Average Latency for (a) simple full file download (uses all available channels), (b) optimal single-layer hashing algorithm (available bandwidth sub-divided into 2 channels between hash and data), (c) non-encoded hierarchical-hashing system and (d) DeltaCast (which also includes the time to decode the erasure data). The 2 hierarchical hashing schemes (c and d) in this case utilise 4 levels of hashing, and 1 data level.

schemes (b) and (c), however, is determined by factors (i), (ii), and (iii), while the latency in scheme (d) is determined by factors (i), (ii), and (iv).

From this Figure we can see that a hierarchical hashing scheme with no encoded data and a single-layer hashing scheme have an average latency that is 2-2.6 times higher than doing a full file download and more than 4 times higher than DeltaCast. The reason for this is that such schemes rely on finding the exact set of hashes and data to be downloaded from the carousel, which can result in very large idle times waiting for the right data to arrive. Observe, however, that while the latency is quite high, these schemes still provide significant bandwidth and power savings compared to a full file download since only a fraction of the time is used in downloading data.

The number of levels in the unencoded hierarchical scheme was set equal to four in this experiment (similar to DeltaCast). We have also tested the performance of such a hierarchical hashing scheme with a higher number of levels and note that the overall latency increases. This is due to the fact that the number of blocks increases as the hierarchy level expands, and thus, it becomes more difficult to find a specific set hashes.

When we compare the performance of DeltaCast to the other schemes we can see that DeltaCast provides significantly lower latencies. DeltaCast requires the same amount of data to be downloaded as a hierarchical scheme that uses non-encoded data. However, with DeltaCast, idle times practically do not exist since any hash or data block will be useful for any receiver as long as it is linearly independent with all the other blocks. A non-linearly independent packet can be generated and received, however since the encoded data field is selected to be very large, the probability of such an event occurring is extremely small.

One drawback of DeltaCast is that erasure hashes and erasure data blocks need to be decoded before they can be used. Decoding time is proportional to the amount of blocks in a given carousel and also proportional to the amount of blocks that need to be decoded. We assume that encoding times are negligible since data can be pre-encoded at the DataCasting head-end. Decoding times are more important since they increase the overall latency and consume resources at the receiver's device. We show the results of an implementation of a fast, software-based, sparse erasure decoder that we have developed. The tests were conducted on a Pentium IV workstation with 256 MBytes RAM which could provide slightly higher performance than a typical small mobile device, however we believe that the results indicate the benefits of utilising erasure coding far outweigh the arguments against. Other more popular Reed-Solomon decoders can also be used for the same purpose.

From Figure 7 we can see that the total latency time for DeltaCast is roughly 2 times lower than the latency of the best scheme. Even when we account for the total decoding time of hashes and data, DeltaCast still offers significant benefits since less data is downloaded and there are no idle waiting times. Decoding times increase as more levels are used since the total number of blocks at lower levels of the hierarchy is significantly higher. However, even for multiple levels of hierarchies, the total decoding time is only few seconds, thus, providing an almost negligible decoding overhead. DeltaCast, trades off latency and downloaded data for decoding time, however, as we have seen in this section the overall penalty is quite low.

3.7 Device Utilisation

Following the experiments from the previous section, we now identify the amount of time that the user's device is utilised, either to download hashes and data, or to decode the file. This time relates to the amount of battery power consumed at the receiver's device. We assume that during the idle times, no power is consumed. However, this assumption provides a lower estimate on the overall device utilisation for those schemes that have to wait long idle times (e.g. hierarchical hashing with no encoded data, and single-layer schemes). During such idle waiting times devices perform periodic attempts to check whether the right hash/data is being carouselled, which could consume non-negligible amounts of battery power. With DeltaCast, on the other hand, all required hashes and data are downloaded consecutively one after the other without random time gaps in between, thus, eliminating the need for periodic data checks.

In table 1 we show the device's activity time for all four schemes described before. The active time total and the average latency values for DeltaCast also include the decoder time. From this table we can see that DeltaCast utilises much fewer resources compared to a full download or a single-layer hashing scheme (approximately 1.7-2.1 times less), which translates into significant battery savings. Compared to a hierarchical hashing scheme with no encoded data, DeltaCast has a 7% higher device utilization due to the extra time to decode data (total of 12 seconds). However, from the same table we can see that by allowing data to be encoded, DeltaCast provides an overall latency that is 4.6 times lower, which clearly outweighs the decoding overhead.

Table 1 also summarises all the results presented up to now, averaged over a large number of Web content channels. Such results include the average latency to update the content, the amount of time that the user's device is utilised, the amount of data downloaded, and the decoding time for four different schemes. From this table we can clearly see that DeltaCast can very quickly reconcile files in a broadcast system while providing significant battery savings.

4 Related Work

Next, we give a brief summary of related work. The rsync protocol [11] is the basis of the very widely used rsync tool. In rsync, receivers use two sets of hashes (one weaker than the other) computed over a fixed block size throughout the file. Such hashes are sent to the source, which then does two passes to identify possible matches. It first compares the fast hash incrementally over the whole file (fixed blocks, utilising a sliding window). If the fast hash matches, then, it tests the more accurate hash. If the accurate hash also matches, then the source only needs to send a hash index to the receiver. The receiver rebuilds the file based on either the hash index or a new data segment sent by the source.

There are a number of theoretical studies of the file synchronisation problem [18], [19], [21], [20]. Within this framework, [22] discusses a relationship between Error Correcting Codes and file synchronisation. Orlitsky and Viswanathan [22] showed a relationship between Error Correcting Codes for noisy channels and file synchronisation that may be on some level related to the DeltaCast erasure-hash approach. A number of authors have studied problems related to identifying disk pages, files, or data records that have been changed or deleted [23], [25], [24].

Hash-based techniques similar to rsync have been explored by the OS and Systems community for purposes such as compression of network traffic [28], distributed file systems [29], distributed backup [30], and web caching [27]. These techniques use string fingerprinting techniques [26] to partition a data stream into blocks. However, several of these techniques usually require variable-size blocks which would complicate the design of a DeltaCast system based on erasure and decomposable hashes.

In [28] efficient erasure encoded data is used to accelerate the delivery of content in multicast-based systems that are prone to errors. We assume that data is reliably delivered to the receivers and focus instead on how to perform efficient file reconciliation. In this regard, [17] provides a framework to reconcile encoded files in peer-to-peer environments.

There has also been some related work that tries to minimise the overhead caused by periodically tuning in to the broadcast channel to find the appropriate data. As such, there are a number of indexing techniques that can be used to improve the performance. [34] [31][32][33]

The work that is closest to ours is [35]. In [35] an erasure-based algorithm is proposed to replace the need for multi-round communication protocols in client-server reconciliation protocols. By estimating the number of required edit operations on the file, and sending sufficient erasure hashes, the receiver is able to regenerate the correct set of hashes at each level. The server must send at least $2k$ erasures, where k is an upper bound on the edit distance. The server continues to send hashes for smaller block sizes until the hash length is equal to the block size, i.e. the original data. In reality this algorithm is not practical since it requires the server to know k accurately before hand. An alternative solution is to have the receiver send sufficient erasures to the source, which then detects how much of the file has changed and sends deltas of the file back to the client. However, both these approaches are better suited for client-server environments with a symmetric communication channel, which does not exist in most DataCasting systems.

5 Summary

We have considered the problem of efficiently reconciling two versions of a file in a broadcast system. Broadcast data systems are becoming very popular as a means for distributing information to a large number of receivers (e.g. news, maps, software, etc.). In a mobile environment where receivers connect at arbitrary times and have power constrained devices, designing a fast reconciliation protocol that minimises battery consumption is quite challenging. This is even more the case in broadcast systems where servers have no information about what type of content or what version resides at each receiver's device and often have to make blind decisions about what content to broadcast.

	DeltaCast			Hierarchical			Single Layer			Simple Download		
Avg. Latency	41.6	120.1	173.4	204.9	583.03	797.93	303.79	305.21	609	0	291.6	291.6
Active Time	41.6	120.1	173.4	41.6	120.1	161.7	303.79	62.9	366.69	0	291.6	291.6
Data Download	31.7	366.8	398.5	31.7	366.8	398.5	83.4	366.8	450.2	0	1780	1780
Decoding Time	0.4	11.6	12	0	0	0	0	0	0	0	0	0

Table 1: Performance Comparison. Time in seconds, Data in KBytes (hashes/data/total).

In this paper we have presented DeltaCast, a practical reconciliation system that combines hierarchical hashing with erasure codes and decomposable hashes. DeltaCast is able to simultaneously update an unlimited number of receivers holding a wide range of file versions. We have evaluated the performance of DeltaCast using a variety of content types, including Web content, and binary files and compared it with several other standard reconciliation techniques. Our results show that DeltaCast is very effective in quickly updating outdated content while conserving scarce power resources in mobile devices. We note also that the Deltacast system is not only useful within the radio broadcast environment, but the technique could also be applied to more general point-to-multipoint systems such as IP Multicast, Overlay networks, Peer-to-peer and Content Distribution Networks within the wired Internet environment. As future work, we plan on producing a stable version of this practical algorithm and use it in a real DataCasting environment to test its performance over a large scale deployment.

References

[1] Q. L. Hu, D. L. Lee, and W.-C. Lee. Performance evaluation of a wireless hierarchical data dissemination system., *In Proceedings of the 5th Annual ACM International Conference on Mobile Computing and Networking (MobiCom99)*, Seattle, WA, August 1999.

[2] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik. Broadcast disks: Data management for asymmetric communications environments., *In Proceedings of ACM SIGMOD Conference on Management of Data*, San Jose, CA, May 1995.

[3] T. Imielinski, S. Viswanathan, and B. R. Badrinath. Data on air Organization and access., *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, May/June 1997.

[4] StarBand., *http://www.starband.com/*

[5] Hughes Network Systems., *http://www.direcway.com/*

[6] Multimedia Broadcast/Multicast System (MBMS), *http://www.3gpp.org/ftp/Specs/html-info/29846.htm*

[7] Broadcast and Multicast Service in cdma2000 Wireless IP Network., *http://www.3gpp2.org/*, October 2003

[8] Digital Audio and Video Broadcasting systems, *http://www.etsi.org/*

[9] First UK user trial of multi-channel TV to mobile phones, *Nokia Press Release*, IBC Amsterdam, 2004.

[10] DirectBand Network. Microsoft Smart Personal Objects Technology (SPOT). *http://www.microsoft.com/resources/spot/*.

[11] A. Tridgell and P. MacKerras, The rsync algorithm, *Technical Report TR-CS-96-05*, Australian National University, June, 1996.

[12] N. Baric and B. Pfitzmann, Collision-free accumulators and fail stop signature schemes without trees, *Advances in Cryptology* EUROCRYPT 97, 1997.

[13] M. Bellare and D. Micciancio, A new paradigm for collision-free hashing: Incrementality at reduced cost, *Advances in Cryptology* EUROCRYPT 97, 1997.

[14] R. Johnson, D. Molnar, D. Song, and D. Wagner, Homomorphic signature schemes, Progress in Cryptology CT-RSA 2002, 2002

[15] T. Schwarz, R. Bowdidge, and W. Burkhard, Low cost comparison of file copies, *Proc. of the 10th Int. Conf. on Distributed Computing Systems*, 1990, pp. 196202.

[16] Zdelta Home Page, *http://cis.poly.edu/zdelta/*

[17] J. Byers and J. Considine, Informed Content Delivery Across Adaptive Overlay Networks, *Proc. of ACM SIGCOMM, August 2002.*

[18] G. Cormode, M. Paterson, S. Sahinalp, and U. Vishkin, Communication complexity of document exchange, *Proc. of the ACM SIAM Symp. on Discrete Algorithms, Jan. 2000.*

[19] G. Cormode, Sequence Distance Embeddings, *Ph.D. thesis*, University of Warwick, January 2003.

[20] A. Orlitsky, Interactive communication of balanced distributions and of correlated files, *SIAM Journal of Discrete Math*, vol. 6, no 4, pp. 548564, 1993.

[21] A. Orlitsky, Worst-case interactive communication II: Two messages are not optimal, *IEEE Transactions on Information Theory* vol. 37, no. 4, pp. 9951005, July 1991.

[22] A. Orlitsky and K. Viswanathan, One-way communication and error-correcting codes, *Proc. of the 2002 IEEE Int. Symp. on Information Theory*, June 2002, p. 394.

[23] Y. Minsky, A. Trachtenberg, and R. Zippel, Set reconciliation with almost optimal communication complexity, *Technical Report TR2000-1813*, Cornell University, 2000.

[24] D. Starobinski, A. Trachtenberg, and S. Agarwal, Efficient PDA synchronization, *IEEE Trans. on Mobile Computing*, 2003.

[25] S. Agarwal, D. Starobinski, and A. Trachtenberg, On the scalability of data synchronization protocols for PDAs and mobile devices, *IEEE Network Magazine*, special issue on Scalability in Communication Networks, July 2002.

[26] R. Karp and M. Rabin, Efficient randomized pattern-matching algorithms, *IBM Journal of Research and Development*, vol. 31 no. 2, pp. 249260, 1987.

[27] S. Rhea, K. Liang, and E. Brewer, Value-based web caching *Proc. of the 12th Int. World Wide Web Conference*, May 2003.

[28] N. Spring and D. Wetherall, A protocol independent technique for eliminating redundant network traffic, *ACM SIGCOMM Conference*, 2000.

[29] A. Muthitacharoen, B. Chen, and D. Mazi'eres, A low bandwidth network file system, *in Proc. of the 18th ACM Symp. on Operating Systems Principles*, October 2001, pp. 174187.

[30] L. Cox, C. Murray, and B. Noble, Pastiche: Making backup cheap and easy, *in Proc. of the 5th Symp. on Operating System Design and Implementation*, December 2002.

[31] M.-S. Chen, K.-L. Wu, and P. S. Yu. "Optimizing index allocation for sequential data broadcasting in wireless mobile computing". *IEEE Transactions on Knowledge and Data Engineering* (TKDE), 15(1):161173, January/February 2003.

[32] T. Imielinski, S. Viswanathan, and B. R. Badrinath. "Data on air Organization and access". *IEEE Transactions on Knowledge and Data Engineering* (TKDE), 9(3):353372, May/June 1997.

[33] N. Shivakumar and S. Venkatasubramanian. "Energy-efficient indexing for information dissemination in wireless systems". *ACM/Baltzer Journal of Mobile Networks and Applications* (MONET), 1(4):433446, December 1996.

[34] J. Xu and W. Lee and X. Tang, "Exponential index: A parameterized distributed indexing scheme for data on air", *In Proceedings of the 2nd ACM/USENIX International Conference on Mobile Systems, Applications, and Services (MobiSys'04)*, Boston, MA, June 2004.

[35] U. Irmak, S. Mihaylov, and Torsten Suel, "Improved Single-Round Protocols for Remote File Synchronization", *In Proceedings of IEEE INFOCOM 2005*, Miami, March, 2005

[36] John Byers, Michael Luby, Michael Mitzenmacher, and Ashutosh Rege, A digital fountain approach to reliable distribution of bulk data, *SIGCOMM*, 1998.

Improving TCP Performance over Wireless Networks with Collaborative Multi-homed Mobile Hosts

Kyu-Han Kim and Kang G. Shin
Department of Electrical Engineering and Computer Science
The University of Michigan, Ann Arbor
{kyuhkim,kgshin}@eecs.umich.edu

Abstract

Multi-homed mobile hosts situated in physical proximity may spontaneously team up to run high-bandwidth applications by pooling their low wireless wide-area network (WWAN) bandwidths together for communication with a remote application server and utilizing their high-bandwidth wireless local-area network (WLAN) in ad-hoc mode for aggregation and distribution of application contents among the participating mobile hosts. In this paper, we first describe the need for such a mobile collaborative community, or a community, in which multi-homed mobile hosts exploit the diversity of WWAN connections to improve a user-perceived bandwidth and network utilization. Then, we show that existing one-to-one communication protocols like TCP suffer significant performance degradation due to frequent packet reordering and heterogeneity of WWAN links in the community.

To address the above TCP problem, we propose a proxy-based inverse multiplexer, called *PRISM*, that enables TCP to efficiently utilize the community members' WWAN connections. PRISM runs at a proxy's network layer as a routing component and stripes each TCP flow over multiple WWAN links by exploiting the transport-layer feedback information. Moreover, it masks variety of adverse effects specific to each WWAN link via intelligent ACK-control mechanism. Finally, PRISM includes a sender-side enhancement of congestion control, enabling TCP to respond correctly to dynamically-changing network states.

We have evaluated the PRISM protocol using both experimentation and *ns-2*-based simulation. Our experimental evaluation has shown PRISM to improve TCP's performance by up to 310% even with two collaborative mobile hosts. Our in-depth simulation study also shows that PRISM delivers a near-optimal aggregated bandwidth in the community formed by heterogeneous mobile hosts, and improves network utilization significantly.

1 Introduction

As wireless networks become omnipresent, mobile users are gaining access to the Internet via a variety of wireless networks. To keep pace with the trend, a mobile host is equipped with multiple wireless network interfaces (e.g., GPRS, IEEE 802.11x, and Bluetooth). Based on such diversity, several researchers attempted to enhance network availability, focusing on concurrent (or alternative) use of multiple wireless technologies available on a host, a mobile user or a designated mobile center [8, 12, 17]. That is, they have attempted to improve network availability within a single individual multi-homed entity.

It is important to note that collaboration among multi-homed mobile hosts significantly improves both user-perceived bandwidth and overall wireless network utilization. Mobile hosts in close proximity can spontaneously form a "community," connected via a high-speed WLAN interface, and share their WWAN link bandwidths with other members in the community. Possible applications of this include (i) contents sharing where each host with same interests receives a subset of contents from an Internet server and share the contents with other hosts, and (ii) bandwidth sharing where one host in a community needs more bandwidth than its own WWAN link for applications like video-on-demand or Hi-Definition TV live cast.

We, therefore, advocate formation of a mobile collaborative community that is a user-initiated network service model and that allows bandwidth sharing/pooling among multi-homed mobile users to make the best of their diverse WWAN links. The mobile community is different from a current WWAN service model, which forces a mobile host to use a single WWAN link at a time, causing capacity, coverage, and hardware limitations. By contrast, the community helps mobile users initiate new virtual WWANs that overcome such limitations by sharing their WWAN interfaces. Moreover, by adopting an inverse multiplexer [9], the mobile community effectively aggregates its members' WWAN bandwidths.

However, the existing transport protocols, such as the Transmission Control Protocol (TCP), are not aware of existence of multiple and parallel intermediate links, and thus, cannot exploit multiple available WWAN links in the community. Even with the help of a multi-path routing protocol, frequent out-of-order packet delivery due to the heterogeneity of WWAN links significantly degrades TCP's performance.

There are several transport-layer solutions for bandwidth aggregation of a *single* multi-homed mobile host such as those in [11–13], but their basic design considers aggregation of the interfaces of only a single host or user, and requires support from the network layer to route traffic to/from a group of multi-homed mobile hosts. Furthermore, the development and deployment of a whole new transport protocol requires significant efforts on both content servers and mobile clients, and also incurs a high com-

putational overhead to resource-scarce mobile hosts.

To solve these problems, we propose a proxy-based inverse multiplexer, called *PRISM*, that enables each TCP connection to utilize the entire community's aggregate bandwidth. As a TCP's complementary protocol, PRISM consists of (i) an inverse multiplexer (PRISM-IMUX) that handles TCP's data and acknowledgment (ACK) traffic at a proxy, and (ii) a new congestion control mechanism (TCP-PRISM) that effectively handles, at a sender side, packet losses over the community's WWAN links.

The first component stripes TCP traffic intelligently over multiple WWAN links using up-to-date link utilization information and each packet's expected time of arrival at a receiver. Also, it masks the effects of out-of-order delivery by identifying spurious duplicate ACKs and resequencing them so that a TCP sender receives correctly-sequenced ACKs.

The second component in PRISM is a sender-side congestion control mechanism (TCP-PRISM) that reduces the loss recovery time and accurately adjusts a congestion window size of TCP by using the loss information provided by PRISM-IMUX. It immediately dis-ambiguates real packet losses from out-of-order deliveries through negative loss information, and reduces the loss recovery time. Its proportional adjustment strategy of the congestion window size further improves link utilization by minimizing the effects of partial congestion on un-congested links.

We evaluate the performance of PRISM using both experimentation and *ns-2*-based simulation. PRISM is implemented as a Linux kernel loadable module and extensively evaluated on a testbed. Our experimental evaluation has shown PRISM to improve TCP's performance by 208% to 310% even with two collaborative mobile hosts with heterogeneous link delays, loss rates and bandwidths. Moreover, our simulation study shows that PRISM effectively reduces the need for packet reordering and delivers the near-optimal aggregated bandwidth in the community that is composed of heterogeneous mobile hosts.

The rest of this paper is organized as follows. Section 2 presents the motivation and the contributions of this work, and Section 3 provides an overview of the PRISM architecture. Sections 4–6 give detailed accounts of PRISM. Section 7 describes our implementation and experimentation experiences. Section 8 evaluates the performance of PRISM using *ns-2*-based simulation. Related work is discussed in Section 9. Finally, Section 10 discusses a few remaining issues with PRISM and concludes the paper.

2 Motivation

We first describe the motivation of a mobile community. Then, we discuss basic functions for the community to work. Finally, we identify the problem of TCP in the community and introduce our approach to the problem.

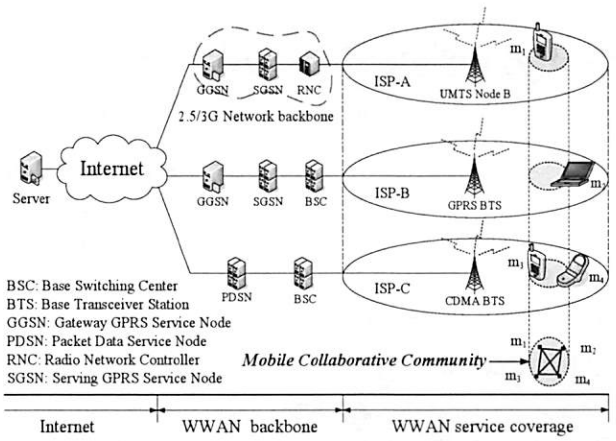


Figure 1: *Target environment.* The environment includes various WWAN network services available and multi-homed mobile hosts equipped with both WWAN and WLAN interfaces. Mobile hosts in a WLAN range form a mobile community and collaborate to simultaneously use multiple WWANs.

2.1 Why a Mobile Community?

Wireless network services become available anywhere and anytime. 2.5G and 3G wide-area cellular networks (e.g., GPRS, UMTS, and CDMA) are being deployed for more bandwidth and wider coverage. Moreover, WLANs (e.g., IEEE 802.11x) can now provide high-speed wireless network services in small areas. At present, different wireless Internet Service Providers (ISPs) are co-located with different network technologies or frequency channels, and end-users are equipped with various wireless interfaces (e.g., WWAN, WLAN, and Bluetooth) and can select the best interface/channel available at a given place/time.

Although there exist many choices (i.e., different ISPs, technologies, and channels) in the current wireless network environment, they are not utilized efficiently due to the current ISP-centric service model. That is, most mobile users should use the same network, technology, or a single frequency channel to get a common service. As a result, they suffer from various service limitations as follows.

L.1 *Capacity limitation:* Mobile users may experience a low data rate from its own ISP while other ISP networks in the same area are idle or under-utilized.

L.2 *Coverage limitation:* A user may find no service nearby from his own ISP while the other ISPs' services are available.

L.3 *Hardware limitation:* A user cannot access a new service through his own interfaces while other users can access the service by their new interfaces.

Let us consider the following scenario to have a feel for the above limitations. Sam is waiting at an airport for his flight, and wants to download his favorite movies to watch during his flight. First, he tries to use his own WWAN interface, but finds that it will take longer than his waiting

time for the flight (capacity limitation). Next, he decides to use his WLAN interface. However, the nearest WLAN hot-spot is too far away for him to return in time for the flight (coverage limitation). Finally, he finds another access network with high capacity, but his device does not support the access network's technology (hardware limitation). Therefore, Sam will not be able to watch the movies. Instead, Sam searches other nearby mobile users who are willing to share their interfaces for certain "rewards." He finds several mobile hosts whose interfaces have capacity, use different frequency channels, or support a high-rate wireless technology like IEEE 802.16. With the help of other mobile hosts, Sam can download movie files in time, and enjoy them during his flight.

To realize a scenario like this, we construct a user-initiated collaborative wireless network model, called a *mobile collaborative community (MC^2)*. As shown in Figure 1, the community is composed of multi-homed mobile hosts in physical proximity. Community members are connected to the Internet via different WWAN ISPs (m_1, m_2) or different channels[1] of the same ISP (m_3, m_4), and forward packets via WLAN interfaces in ad-hoc mode.

2.2 How Does a Mobile Community Work?

As basic building blocks, a mobile community has three functions: collaboration, multiplexing, and indirection.

2.2.1 Collaboration Among Mobile Hosts

The mobile community requires users to collaborate by sharing/pooling their communication channels. However, what are the incentives for users to collaborate? When only one host or a small set of members want to receive the content at others' expenses, will the other members be willing to contribute their bandwidth to enable the small set of members to achieve statistical multiplexing gains?

A somewhat related debate is underway with regard to "forwarding incentives" in ad hoc network routing [7, 18, 23]. In ad hoc networks, the communication between end-points outside of the radio transmission range relies on intermediate nodes on the path to forward packets for them. Some researchers suggest use of credit-based, or reputation-based, schemes to stimulate cooperation [7, 14]. Game-theoretic arguments have been used to show that collaboration on packet forwarding among all participating nodes will yield maximum network throughput.

Forwarding in ad hoc networks, however, is somewhat different from the collaboration we consider here. In ad hoc networks, nodes rely on each other to communicate amongst themselves. In a mobile community, nodes rely on each other not for basic connectivity, but for performance improvements. As we will see in Section 3, a node completely controls access to its shared communication resources, and revokes access if its communication needs are not met by the community. Ultimately, it is the ability to opt-in to achieve better performance and the ability to opt-out when necessary, making link sharing a viable option.

2.2.2 Multiplexing

Given shared links, how can the mobile community aggregate link bandwidths for a higher throughput? An inverse-multiplexer is a popular approach that aggregates individual links to form a virtual high-rate link [9]. For example, an inverse multiplexer stripes the traffic from a server over multiple wireless links of the community members, each of which then forwards the traffic to the receiver. Finally, the forwarded packets are merged in the receiver at the aggregate rate.

Then, an issue is where to put the inverse multiplexer. The inverse multiplexer can be placed at a performance-enhancing proxy by a network access provider, a wireless telecommunication service provider, or a content distribution network operator. Furthermore, it can be placed in a network layer as one routing component with an efficient traffic filtering function as in the Network Address Translation (NAT) service. On the other hand, the inverse multiplexer might run as an application like in an overlay network. However, multiplexing inherently requires responsive network state information, and additional packet processing overheads at the application layer limit the performance of the inverse multiplexer [12].

2.2.3 Indirection

Traffic from an inverse multiplexer to community members is tunneled via Generic Routing Encapsulation (GRE) [10]. The inverse multiplexer encapsulates the traffic via GRE and routes it to the community members. Each member de-capsulates the tunneled traffic, upon its reception, and forwards it to a destination via WLAN. Since the destination is oblivious to which member forwarded the data packets, no additional data reassembly functionality is required at the receiver. Furthermore, because GRE tunneling is supported by most operating systems (e.g., Linux, FreeBSD, the Windows), no system modification of mobile hosts is required.

2.3 Challenges in MC^2's Use of TCP

Our primary contribution in this paper is that in an MC^2, we enable *one-to-many-to-one* communication for a TCP connection to achieve high-speed Internet access. While traditional one-to-one communication of TCP limits its bandwidth to a single link's capacity, in an MC^2, we enable a TCP connection to achieve the members' aggregate bandwidth by inverse-multiplexing its packets over all available links.

In this communication model, however, we encounter several challenges. First, *scheduling* traffic over wireless links requires exact link state information such as data

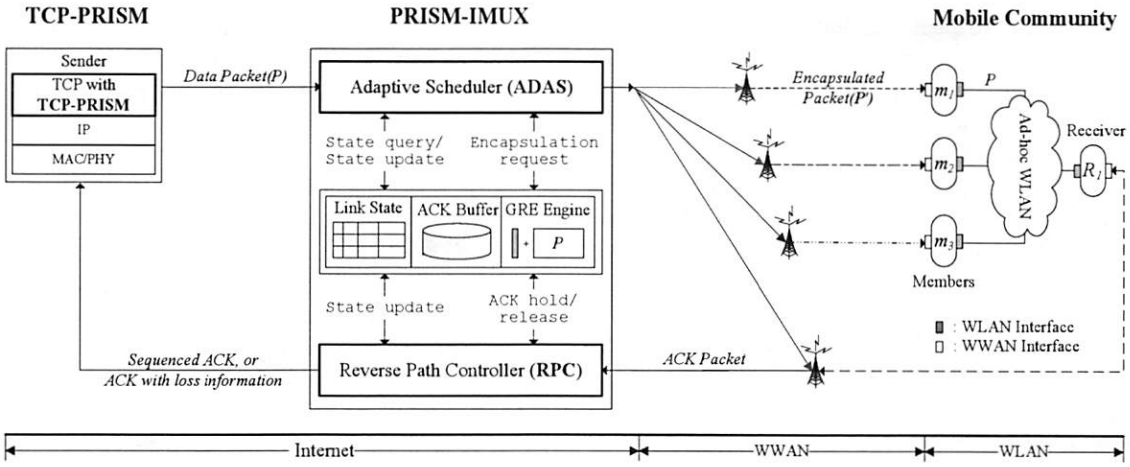


Figure 2: *PRISM architecture.* PRISM consists of an inverse multiplexer at the proxy (PRISM-IMUX) and a sender-side congestion control mechanism (TCP-PRISM). PRISM-IMUX captures each data packet in the middle of a sender and a receiver. After selecting the best WWAN link for each packet's next hop using the Adaptive Scheduler (ADAS), PRISM-IMUX forwards the packet to the WWAN link after encapsulating it via GRE. The encapsulated packet arrives at a community member via the WWAN link, and the member de-capsulates and forwards the packet to the receiver. Next, the receiver receives and processes the packet as a normal packet, and then returns an ACK packet to the sender. PRISM-IMUX again captures the in-transit ACK and decides whether the ACK is a spurious duplicate ACK or not, using the Reverse Path Controller (RPC). If PRISM-IMUX detects packet losses from duplicate ACKs, it releases the ACK with loss information to the sender, and TCP-PRISM at the sender uses the delivered loss information for fast loss recovery. If there is no loss, PRISM-IMUX holds or releases the ACK in a sequenced order.

rate and delay, which varies with time and is usually expensive to obtain in mobile environments. Second, because WWAN links suffer from high and variable round trip times (RTTs), burstiness and outages, the large number of *out-of-order* packet deliveries, which generate spurious duplicate ACKs, degrade TCP's end-to-end performance significantly. Finally, TCP's *congestion control* mechanism does not fully utilize multiple links' bandwidths because it interprets a packet loss as the overall links' congestion, making over-reduction of its congestion window size. Also, frequent spurious duplicate ACKs with positive ACKs cause the sender to delay loss detection/recovery.

2.4 Improving TCP Performance in an MC^2

To overcome the above challenges, we propose a proxy-based inverse multiplexer, called *PRISM*, that effectively aggregates members' WWAN links bandwidths for a TCP connection. Specifically, we

- **C.1** devise an adaptive scheduling mechanism that stripes traffic with the least cost while maintaining full links utilization;
- **C.2** construct an ACK-control mechanism that effectively shields the effects of out-of-order delivery without sacrificing end-to-end performance; and
- **C.3** propose a new congestion-control mechanism that is a sender-side optimization technique and that improves links utilization by expediting loss recovery.

The rest of this paper provides a detailed account of PRISM. The following assumptions are made for the basic design of PRISM and mobile community: (i) each mobile host has multiple (especially WWAN and WLAN) interfaces that can be used simultaneously for a single application connection; (ii) a mobile community is formed via an application-layer daemon; (iii) GRE is enabled as a default; and (iv) all hosts support TCP-SACK.

3 PRISM Architecture

Figure 2 depicts the architectural overview of PRISM and its operational environment. PRISM consists of a network-layer inverse multiplexer (PRISM-IMUX) at the proxy and a network-assisted congestion-control mechanism (TCP-PRISM) at the sender side. PRISM interacts with a mobile community that is composed of multi-homed mobile hosts.

3.1 PRISM-IMUX

PRISM-IMUX is the routing component in a proxy that handles both the forward (data) and backward (ACKs) traffic of a TCP connection using up-to-date wireless links state information. PRISM-IMUX captures data traffic from a sender in the proxy's network layer, and decides the best WWAN link as a next-hop through the Adaptive Scheduler (ADAS). It also captures and controls ACK packets to shield the adverse effects of striping over multiple WWAN links via the Reverse Path Controller (RPC). Finally, PRISM-IMUX maintains a WWAN links' state table, has a buffer that temporarily stores ACKs which need to be re-sequenced, and supports GRE for indirection. We will detail ADAS in Section 4, and RPC in Section 5.

3.2 TCP-PRISM

TCP-PRISM is a new sender-side congestion-control mechanism that works with PRISM-IMUX to expedite loss recovery, thus improving network utilization. TCP-PRISM reduces the loss recovery time via using the *negative* ACK information shipped by RPC at the proxy to detect a packet loss. Also, it adjusts the congestion window size according to the congested link bandwidth only, thus preventing waste of uncongested links' bandwidth. We will detail this in Section 6.

3.3 Mobile Community

A mobile community is formed voluntarily and incrementally. When a new mobile node wants to join an existing mobile community, it first searches for communities nearby using the Service Location Protocol [19]. After determining the community of most interest to itself, the mobile node/host joins the community and works as either a relay node or a receiver. The node receives packets from PRISM-IMUX via its WWAN link, and forwards packets to the receiver, through its WLAN interface in ad-hoc mode. Or, the node receives packets via multiple community members' WWAN links, and sends ACKs to the sender through one of the WWAN links.

4 Scheduling Wireless Links: ADAS

4.1 Overview

Scheduling TCP packets over heterogeneous wireless links requires exact link state information for a receiver to achieve the optimal aggregate bandwidth, and obtaining the information is expensive, especially in mobile environments due to dynamic traffic rate and wireless links' dynamics. As shown in Figure 3, the typical TCP traffic rate fluctuates as a result of its congestion and flow control. Similarly, the output rate varies due to the heterogeneity of wireless links and/or the processing power of each member. Although it is possible to measure a channel's condition and report it to the proxy, frequent changes in the channel condition will incur significant overheads (e.g., message processing overhead and transmission power consumption) to resource-limited mobile hosts.

ADAS is a new packet-scheduling algorithm that is adaptive to dynamic input/output rates, and is flexible enough to deal with the lack of rate information. ADAS maintains up-to-date links state which is inexpensively obtained by RPC (see Section 5), and adaptively schedules traffic over the best available links using the state information. Also, it uses packets' expected arrival times over each link not only to reduce out-of-order packet deliveries, but also to increase the end-to-end throughput. Finally, ADAS adaptively reacts to a link's congestion via a TCP's AIMD-like traffic control mechanism.

Algorithm 1 ADAS_Scheduling ($Packet$)

$hRTT_i$	RTT from a proxy to a receiver over WWAN $link_i$
l_{next}	Next link which a packet is assigned to
N_i	The number of in-flight packets on $link_i$
$\mathbb{S}$	A set of community members' WWAN links
$\mathbb{S}_{next}$	A set of links with the minimum utilization
U_i	Link utilization of $link_i$

1: **if** $Packet$ is a retransmission **then**
2: *// Rule.1: send the retransmission over a fast link*
3: l_{next} is the link with minimum $hRTT$ in $\mathbb{S}$
4: **else**
5: *// Choose links with the most available bandwidth*
6: $\mathbb{S}_{next}$ = {l_i: links with minimum U_i from $\mathbb{S}$}
7: **if** $|\mathbb{S}_{next}| == 1$ **then**
8: *// Rule.2: use link utilization (U_i)*
9: l_{next} is the link (in $\mathbb{S}$) with minimum U_i
10: **else**
11: *// Rule.3: use an expected time of arrival and U_i*
12: l_{next} is the link with minimum $hRTT$ in $\mathbb{S}_{next}$
13: **end if**
14: **end if**
15: Update N_i, U_i for l_{next} *// update channel information*
16: Update $\mathbb{S}$ *// apply new U_i to the set $\mathbb{S}$*
17: **return** l_{next}

4.2 Algorithm

ADAS consists of three scheduling rules, and dynamic link-weight adjustment mechanism. Algorithm 1 describes ADAS's scheduling rules. *Rule.1* is to give retransmissions priority. Under *Rule.2*, ADAS chooses the link with the most available bandwidth by using link utilization (U_i). Under *Rule.3*, if there are more than two links with the same utilization, then ADAS picks the link that has the smallest expected arrival time ($hRTT$).

4.2.1 Link Utilization (U_i)

Link utilization enables ADAS to utilize multiple links fairly so as to maximize an aggregated bandwidth. U_i is derived from the Weighted Round Robin (WRR) scheduling for its fairness. WRR divides the time into a round, in each of which packets are assigned to a link based on its proportional bandwidth (or weight), and thus, all links are utilized fairly. Likewise, ADAS uses the link-weight for fair link utilization, thus achieving long-term fairness as WRR does.

However, ADAS uses a different definition of link utilization; *while WRR keeps track of how many packets have been scheduled so far on each outgoing link, ADAS considers how many packets are currently in-flight on the link.* Because existing static scheduling algorithms (e.g., WRR) assume accurate and stable links state information, the link utilization based on the algorithm might not be accurate due to network dynamics. ADAS exploits the actual number of in-flight packets, which can be derived from scheduled packets' information and ACK-control information, and which automatically reflect unexpected delay or loss

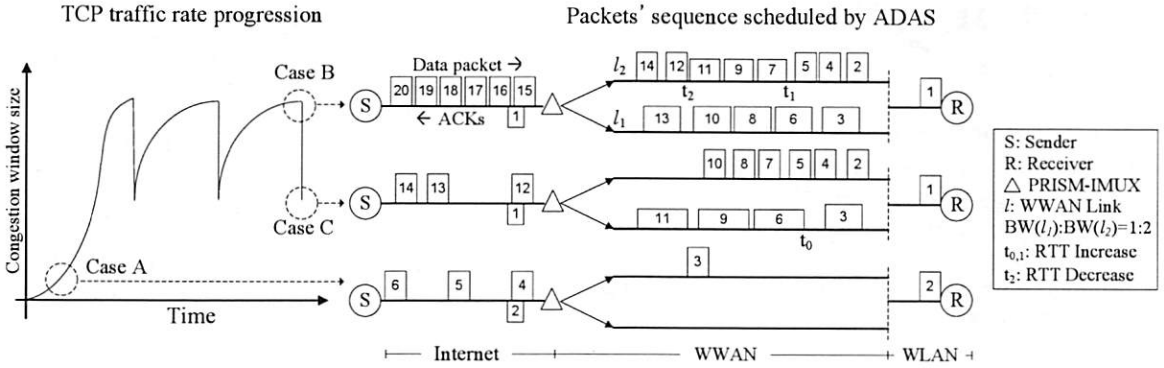


Figure 3: *Three ADAS scheduling snapshots in different input/output rates.* The left graph shows the fluctuation of input rate from a TCP sender. The right three wide figures are snapshots of ADAS scheduling in different input/output rates. Case A shows Rule.2 (based on link utilization), Case B shows Rule.3 (based on RTT as well as U_i), and Case C shows dynamic weight adjustment.

of a link, in order to determine the utilization of each link. Therefore, we define "link utilization" as $U_i = \left\lfloor \frac{N_i}{W_i} \right\rfloor$, where N_i is the number of in-flight packets over link i, and W_i (link weight) is the ratio of the link bandwidth to the least common denominator/factor of all links' bandwidths.

Let's consider Case A in Figure 3 to see the effectiveness of U. The ratio of the weight of link ℓ_1 to that of link ℓ_2 is assumed to be 1:2. ADAS schedules the third packet (p_3) on ℓ_2 because when p_3 arrives at the proxy, ADAS knows from an ACK packet (a_2) that p_2 has left ℓ_2, so ℓ_2 still has more available bandwidth than ℓ_1. In case of WRR, it assigns p_3 to ℓ_1 because the quantum of ℓ_2 has already exhausted by p_1 and p_2, wasting available bandwidth of ℓ_2.

4.2.2 Expected Arrival Time ($hRTT$)

ADAS uses expected arrival time (half RTT or $hRTT$) along with U to further improve overall link utilization and minimize the need for packet reordering. When more than one link ($\mathbb{S}$) have the same lowest link utilization, ADAS selects the link that has the smallest expected arrival time in that subset of links (Rule.3). Due to a WWAN link's varying transmission delay or forwarding nodes' unexpected processing delay, links with similar utilization might experience different short-term rates or delay fluctuations which might not be immediately reflected into U. Using $hRTT$ ensures that ADAS transmits packets on the fastest link in a greedy fashion, and thus, not only increases the overall short-term links utilization, but also reduce out-of-order packet deliveries at a receiver.

Let's consider Case B in Figure 3 to illustrate the effectiveness of $hRTT$. Until p_7, ADAS has packets scheduled on each link with the same sequence as WRR does. However, at t_1, $hRTT$ of ℓ_2 increases and at the point of scheduling p_8, the expected arrival time of p_8 via ℓ_2 becomes longer than that via ℓ_1. Besides, since the U values of both links are same, ADAS schedules p_8 on ℓ_1. If the packet is scheduled on ℓ_2 as WRR does, then p_8 might arrive later than p_9, and ℓ_1 could waste its bandwidth until the transmission of p_9 begins.

4.2.3 Dynamic Link-Weight Adjustment

ADAS adapts to each link's congestion without separate links state probing or congestion notification messages from networks by dynamically adjusting the congested link's weight. ADAS uses the loss information obtained by RPC (to be explained in the next section), and adjusts the link weight to approximate its instantaneous bandwidth by adopting the TCP's Additive Increase and Multiplicative Decrease (AIMD) strategy. If the link experiences congestion, ADAS cuts the congested link's weight by half. Subsequently, the link's U_i becomes larger, and new packets are not assigned to the link until it recovers from the congestion. This link weight increases additively each time an ACK arrives on that link, without exceeding the original weight. This way, ADAS adaptively reacts to partial links' congestion and controls the amount of traffic to be allocated to each link without requiring expensive instantaneous bandwidth information.

Case C in Figure 3 depicts ADAS's reaction to both delay fluctuations and packet losses. When p_6 is scheduled at t_0, ℓ_1 experiences an increased $hRTT$. However, ADAS schedules p_6 on ℓ_1 based on U to maintain maximum network utilization even though it might cause packet reordering. On the other hand, right before scheduling p_{11}, ADAS identifies a packet loss on ℓ_2. It adaptively reduces the ℓ_2's weight, and assigns p_{11} to ℓ_1 based on the new computed link utilization.

4.3 Complexity

The main computational complexity of ADAS comes from the sorting of links to find the best link. Since ADAS uses an ordered list, it requires $O(\log n)$ time complexity where n is the number of available links. Usually, n is less than 10, so its overhead is not significant. ADAS requires constant space complexity. ADAS maintains a link-state ta-

ble as shown in Figure 2. It independently stores per-link information which includes only four variables (i.e., U_i, $hRTT_i$, W_i and N_i).

5 Handling Spurious Duplicate ACKs: RPC

5.1 Overview

Even though ADAS tries to minimize the need for packet reordering, data packets are sometimes scheduled out-of-sequence explicitly to fully utilize networks (e.g., Case C in Figure 3). Moreover, due to the delay fluctuations resulting from the aggressive local retransmission mechanism of 3G networks or a community member's processing delay, there could be unexpected out-of-order packets. In both cases, a receiver blindly generates duplicate ACKs, which we call 'spurious' duplicate ACKs, as a false sign of link congestion, and these ACKs, unless handled properly, significantly degrade TCP performance.

The Reverse Path Controller (RPC) is an intelligent ACK-control mechanism that hides the adverse effects of out-of-order packet deliveries to the receiver. RPC exploits TCP's control information which is carried by ACKs, to determine the meaning of duplicate ACKs and correct them, if necessary. Moreover, along with the scheduling history, RPC also infers the link condition such as its packet loss, delay, and rate. Finally, because RPC maintains each link's state information (including loss and instantaneous capacity), it provides such information to the sender's congestion-control mechanism so as to prevent one pipe from stalling other uncongested pipes, thus enhancing network utilization.

5.2 Algorithm

RPC consists of three different mechanisms: ACK identification, ACK re-sequencing, and loss detection. RPC accepts ACKs as input, and then holds or releases them based on the above three mechanisms. RPC first identifies the meaning of an arrived ACK. Then, it decides whether this ACK is normal or spurious. Finally, it differentiates duplicate ACKs caused by real packet losses from spurious duplicate ACKs, and detects any packet loss.

5.2.1 ACK Identification

In order to determine the meaning of ACKs, this mechanism identifies the sequence number of a data packet that actually arrives at the receiver and causes an ACK to be generated. Assuming that the receiver supports the TCP-SACK mechanism, RPC traces the meta-state of the receiver buffer through SACK blocks and a cumulative ACK number, and finds the latest updated sequence number of the receiver buffer via the newly-arrived ACK. Because TCP-SACK conveys information of up to three data blocks when there are holes in the receiver buffer, and its first block[2] contains the sequence number of the recently-arrived data packet [15], RPC can infer the state of the receiver's buffer via following two ways.

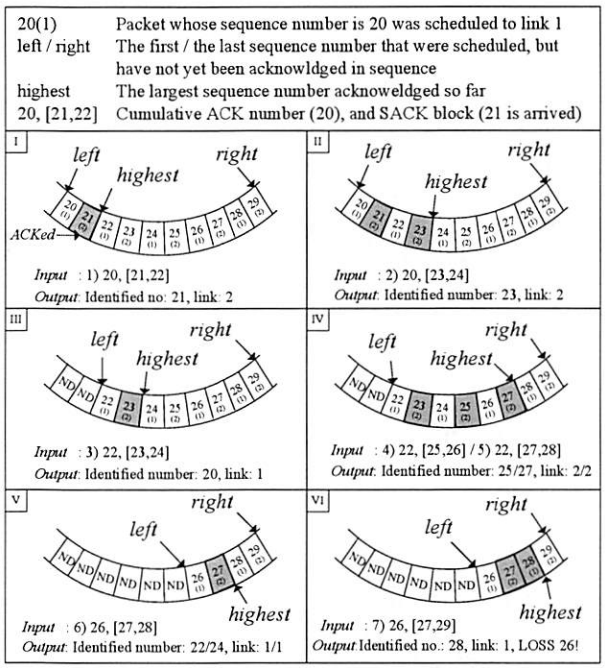


Figure 4: *Snapshots for ACK identification and loss detection mechanism in RPC.* RPC identifies the meaning of each ACK through SACK blocks (Snapshot I, II, and IV) or cumulative ACK numbers (Snapshot III, V). Also, it detects packet losses using identified sequence numbers and scheduling history (Snapshot VI).

A.1 *SACK block matching:* If an ACK delivers SACK information, RPC simply matches the SACK block(s) with the meta-state buffer and finds sequence number(s) that is newly covered by this SACK block.

A.2 *Cumulative ACK number scanning:* If an ACK sequence number is greater than the meta-buffer's cumulative sequence number, RPC scans a region between the two numbers, and finds the sequence number(s) that has not been covered before.

Figure 4 shows a series of snapshots that describe the two schemes of identifying a sequence number. For example, the snapshot I, II, and IV show the SACK block matching scheme. The snapshot III and V illustrate how cumulative ACK numbers are scanned. Each snapshot contains the circular buffer representing the meta-state of the receiver buffer.

5.2.2 ACK Re-sequencing

After identifying the meaning of ACKs, RPC determines whether to release this ACK to the sender or to hold it for re-sequencing based on Algorithm 2. If the identified sequence number proceeds towards a new unACKed sequence number, RPC starts releasing ACKs-on-hold including the one just arrived.

If arrived ACK packets are duplicates (line 8), then RPC re-sequences them in two different ways. First, if there

Algorithm 2 ACK re-sequencing ($Packet$)

```
1: // N is the size of circular ACK re-sequencing buffer
2: wSeqNum = (cumulative ACK sequence number) mod N
3: if wSeqNum > left then
4:    left = wSeqNum        // in-sequence, new ACK
5:    put ACK at the end of the re-sequencing buffer
6:    release held ACK(s) up to left point with a peak rate
7: end if
8: if wSeqNum == left then
9:    switch RPC state      // duplicate ACK
10:       case NORMAL        // re-sequencing
11:          hold ACK with an identified number, break
12:       case LOSS          // no re-sequencing
13:          release ACK to a sender, break
14:   end switch
15: end if
```

is not any congested link (i.e., all links are in NORMAL state), then RPC holds the ACK packet in the slot of the wrapped-around re-sequence number ($wSeqNum$) in the circular ACK buffer. Since RPC knows the meaning of an ACK, it corrects the cumulative ACK sequence number with the identified number of the ACK packet and stores it in the buffer. Second, if RPC is in the LOSS state, it releases ACKs-on-hold in their original form because duplicate ACKs have really resulted from packet loss(es), and because released duplicate ACKs can help the sender calculate the number of packets that have left the network.

5.2.3 Loss Detection

The remaining questions on the ACK re-sequencing mechanism are how to detect packet losses from congestion, and how to differentiate out-of-order packet arrivals from real packet losses. Assuming that packets scheduled on a link are delivered to a receiver in order, RPC detects packet losses if there are holes that are not sequentially acknowledged in a list of scheduled packets on the link. The snapshot VI shows an example of loss detection of RPC. Since packets 26, 28 were sent back-to-back via link 1, RPC determines, from the arrival of ACK 28, that packet 26 is lost. This is different from the loss detection mechanism of TCP whose duplicate ACKs' threshold is 3. However, a more sensitive reaction on each link is desirable since it helps all connections avoid disrupting one another. Moreover, any threshold can be set based on the network characteristics.

5.3 Complexity

RPC's complexity heavily relies on the number of duplicate ACKs. When there is no duplicate ACKs, RPC does not incur any overhead except for updating link-state variables (U, N). However, if there are duplicate ACKs resulting from either out-of-order delivery or a packet loss, then RPC needs to figure out (compute) the ACK's sequence number and space to re-sequence ACKs.

Given duplicate ACKs, RPC only requires constant time complexity and 3KB space complexity per connection in the worst case. The most computation-intensive mechanism in RPC is ACK identification which requires extensive sequence comparison. However, this overhead can be minimized using such optimization techniques as a bit-operation, and thus requires constant time complexity. RPC may have to store all ACKs of a flow in the worst case. Since the number of ACKs is limited by the number of outstanding packets in the network, $\frac{BDP}{MSS} \times S_{ACK}$ is the maximum required ACK re-sequencing buffer size. For example, assuming that an aggregated bandwidth and average RTT are 5 Mbps and 120 ms, respectively, and MSS is 1.5 KB and the size of ACK (S_{ACK}) is 60 bytes, a maximum space requirement is 3 KB.

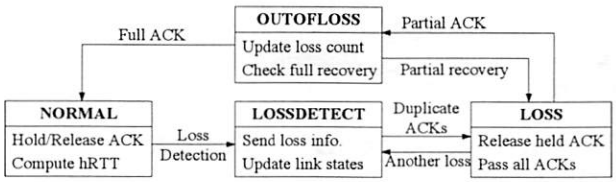


Figure 5: *State machine of RPC.* Boxes with capital letters indicate states of RPC, and boxes with small letters list operations in each state.

6 Expediting Loss Recovery: TCP-PRISM

6.1 Overview

Along with ACK re-sequencing and loss detection, fast recovery from loss(es) and appropriate congestion control at the sender side are critical to the overall performance of a TCP connection. Although many congestion control mechanisms, such as Reno, New-Reno and SACK, have been proposed for a single path congestion control, they are not optimized for multiple paths mainly for the following two reasons. First, TCP's positive ACK mechanism (e.g., SACK block) consumes more time to detect/recover packet loss or out-of-order delivery from multiple heterogeneous paths, resulting in frequent timeouts. Second, they over-reduce the window size upon congestion of one of multiple paths, reducing the overall links utilization.

PRISM addresses these problems by using two mechanisms. The first mechanism provides exact loss/congestion information in a negative form to a TCP sender. The second is a sender-side congestion control mechanism (TCP-PRISM), which understands negative ACK information from networks and expedites loss recovery upon congestion of a link in one of multiple paths.

6.2 Algorithm

This algorithm is invoked by RPC and the sender-side TCP when there is a packet loss(es). On detection of any packet loss, RPC ships loss information on ACKs. Using this delivered information at the sender, its congestion control mechanism quickly reacts to packet losses.

6.2.1 Delivery of Loss Information

Figure 5 shows the state machine of RPC that describes loss information delivery in each state. In NORMAL and OUTOFLOSS states, RPC updates state variables as described in Section 5. In LOSSDETECT state, RPC sends the loss information to the sender, and switches to LOSS state. RPC in LOSS state releases all duplicate ACKs until all losses are recovered.

RPC provides loss information to the sender that includes: (i) which data packet is lost, (ii) which channel is congested to adjust the congestion window size, and (iii) how many packets have left the network. Once a packet loss is detected, RPC sends the lost packet's sequence number to the sender in the form of negative ACK. In addition, RPC ships the congested link's bandwidth information which can be computed by $p = 1 - \frac{B_i}{2\sum_{j=1}^{n} B_j}$ where i is the congested channel ID, B_j the bandwidth of channel j, and n the total number of active channels. Finally, after sending loss information, RPC begins releasing ACKs-on-hold, if any, so that the sender can calculate the exact in-flight packet number, inflate the congestion window size, and send more data packets via other uncongested links.

6.2.2 Congestion Control Mechanism

TCP-PRISM makes two major enhancements of existing congestion control mechanisms. First, it reduces the fast retransmit time given partial link's congestion by using the loss information delivered from the proxy. TCP-PRISM just extracts lost packets' sequence numbers and retransmits the corresponding data packets immediately. It does not wait for more duplicate ACKs, nor does retransmit all packets which are ambiguously believed to have been lost.

Second, it makes fast recovery accurately react to congestion, and thus, improves network utilization. TCP-PRISM reduces the congestion window size only by the proportion (p) of congested link's bandwidth over total bandwidth—we call this adjustment *Additive Increase and Proportional Decrease*. This adjusted window size allows the sender to transmit more data via uncongested links. If there are other congested links, TCP-PRISM performs the same procedure as the first reduction step. Other than the above two enhancements, TCP-PRISM works exactly same as the way vanilla-TCP does.

6.3 Complexity

The complexity of TCP-PRISM is lower than that of the standard TCP-SACK. TCP-SACK's scoreboard mechanism maintains positive ACK information from a receiver and then identifies lost segments. It requires an extensive search to construct up-to-date blocks whenever a SACK block is delivered, and hence, may require more memory space. In contrast, TCP-PRISM uses a simplified version of scoreboard, which only maintains a list of lost packets from the negative loss information.

7 Implementation

We have implemented, and experimented with, PRISM. In this section, we first describe the implementation details of each PRISM component. Then, we describe our testbed setup and present the experimental results.

7.1 Implementation Details

7.1.1 PRISM-IMUX

PRISM-IMUX is implemented as a loadable kernel module in a network layer using Netfilter [1]. Netfilter provides a hook for packet filtering at the network layer, allowing users to dynamically register or un-register any filter. Thus, PRISM-IMUX is implemented as a filter with a back-end agent which includes such mechanisms as ADAS, RPC, and link's state maintenance.

Within the network layer, there are three places for the PRISM-IMUX filter to be registered (at entrance, NF_IP_PRE_ROUTING; in the middle, NF_IP_LOCAL_OUT; and at exit, NF_IP_POST_ROUTING), and it is registered at the layer's exit because this placement minimizes the number of functions that PRISM-IMUX should incorporate, and avoids any need for system modification. When it transmits multiple packets stored in its buffer, PRISM-IMUX can make a direct call to an interface function of the link layer, and thus, it need not go through all the remaining network-layer functions.

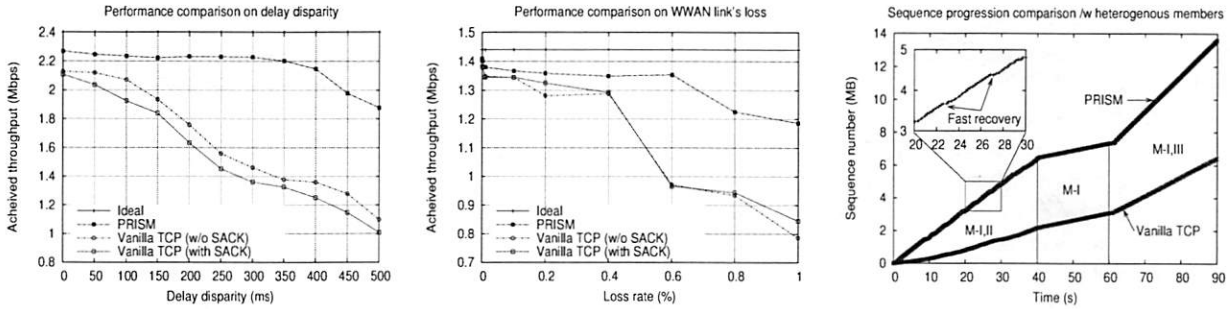
Finally, there is a case where the agent of the filter needs to store packets, and thus stop the remaining packet processing chain in the network layer. There are two options (NF_DROP and NF_STOLEN) from Netfilter to silently store a packet, and PRISM-IMUX uses NF_STOLEN because it does not incur any overhead such as buffer-copy which is required in NF_DROP.

7.1.2 TCP-PRISM

We implemented TCP-PRISM in a Linux kernel-2.4's TCP protocol stack, and deployed it in a server. As described in Section 6, TCP-PRISM is a simplified version of TCP-SACK and easily extensible from TCP-SACK implementation. TCP-SACK maintains sack-tag information, which is initially cleared, and becomes "SACKED" when the corresponding sack information arrives. However, determining an un-sacked packet as a lost packet is still not an easy problem, and thus, TCP-SACK has both a heuristic decision algorithm and an undo algorithm to fix any incorrect decision. We modified this sack-tag mechanism so that the exact loss information provided by PRISM-IMUX is immediately reflected into the sack-tag information.

7.2 Testbed Setup

To evaluate our PRISM implementation, we have built a testbed that is composed of an Internet infrastructure, and a mobile community.



(a) Effects of delay disparity

(b) Effects of loss-rate difference

(c) Effects of link heterogeneity

Figure 6: *Experimental results.* We run PRISM in our testbed and compare its performance with that of vanilla-TCP. In the experiment (a), we measure the throughput achieved by both PRISM and vanilla-TCP while increasing the bandwidth disparity of WWAN links. In the experiment (b), we also measure the throughput while increasing WWAN link's loss rates. In the experiment (c), we conduct the same experiment in a community consisting of heterogeneous members. Member 1 (M-I) with a slow link (360 Kbps) and member 2 (M-II) with fast but a lossy link (1080 Kbps, 0.6%) initially collaborate, then M-II leaves the community (at 40s), and member 3 (M-III) with a fast link (1800 Kbps) joins (at 60s) and collaborates with M-I.

For the Internet infrastructure, we use one server (Pentium-IV 1.64 GHz CPU with 512 MB memory), one proxy, one WWAN emulator (both are a Pentium-III 865 MHz CPU with 256 MB memory), and one Ethernet switch. The server and the proxy have TCP-PRISM and PRISM-IMUX installed, respectively. The emulator has NISTnet [2] to emulate WWAN networks (the proxy and the emulator each have two Ethernet interfaces to construct different networks). The Ethernet switch works as WWAN access networks and splits traffic from the emulator to each community member. The server, the proxy, the emulator, and the switch in a row are connected via 100 Mbps Ethernet cables between successive components.

For the mobile community, we use three Dell latitude laptops (Pentium-III 865 MHz CPU with 256 MB memory) which have both built-in Ethernet interfaces (Realtek) and IEEE 802.11b interfaces (Orinoco). Each Ethernet interface is connected to Ethernet switch's 100 Mbps cables and is used as WWAN links. A WLAN interface in ad-hoc mode is used for communication within the community.

All machines in the testbed use Redhat 9.0, and an ftp application between a server and a receiver is used to measure end-to-end throughput by transferring a 14MB file.

7.3 Experimental Results

7.3.1 Effects of delay disparity

We evaluated the performance tolerance of PRISM to the WWAN links delay disparity. We use two community members which have different bandwidths (1800, 600 Kbps) but initially have the same link delay, 500ms (average delay from the UMTS trace with the packet size of 1.4 KB).[3] While increasing one link's delay up to 1000 ms in increments of 50 ms, we measure end-to-end throughput. For better comparison, we also run vanilla-TCP with and without SACK.

PRISM effectively masks the delay disparity of WWAN links and provides an aggregated bandwidth through RPC's re-sequencing mechanism. Figure 6(a) shows that PRISM achieves 95% throughput of total aggregate link capacity when the delay disparity is less than 400 ms. Beyond that point, it shows a little degradation because of deep-buffering for increasing duplicate ACKs. Vanilla-TCP suffers significant performance degradation due to spurious duplicate ACKs. Furthermore, vanilla-TCP with SACK shows worse performance than that without SACK because detailed SACK information delivered to a sender causes significant false retransmissions.

7.3.2 Effects of loss-rate disparity

We measured the performance tolerance of PRISM to the WWAN links loss-rate difference. In the community with two members (whose bandwidths are 1080, 360 Kbps), we fix the link delay of both members at 300 ms (average delay from the UMTS trace with the packet size of 1 KB) and measure end-to-end throughput as we vary the loss-rate from 0.001% to 1% of the second member (1% is a typical maximum loss-rate of WWAN links).

The fast-recovery mechanism in PRISM indeed expedites loss recovery and increases link utilization even at a high loss-rate. As shown in Figure 6(b), PRISM provides 94% throughput of the total links capacity when loss rates are less than 0.8%. At the point of 0.8% or higher, PRISM's throughput decreases because the achievable link throughput also degrades due to frequent packet losses. Vanilla-TCP, however, experiences a severer performance degradation. Even though it shows relatively good performance (i.e., 90%) at a low loss rate, vanilla-TCP immediately shows degraded performance as the

loss-rate increases because of the long loss-recovery time for one congested link, blocking the uncongested link.

7.3.3 Effects of link heterogeneity

We evaluated the performance gains of PRISM even with heterogeneous community members. We construct a mobile community that consists of three members, all having different WWAN link characteristics (bandwidth, delay, and loss rate) as follows: Member 1 (M-I) has a reliable but slow link (360 Kbps, 300 ms, 0%); member 2 (M-II) has a fast but unreliable link (1080 Kbps, 100 ms, 0.6%); and member 3 (M-III) has a faster link (1800 Kbps, 100 ms, 0%) than others, but its bandwidth difference from M-I's is large (5 times). Initially, M-I and M-II collaborate until 40 seconds, but face different delays and loss-rates. Then, M-II leaves the community (at 40s). At 60s, M-III joins the community and collaborates with M-I, but they have a large bandwidth disparity.

PRISM achieves the aggregated bandwidth of all WWAN links even in case of heterogeneous link characteristics. Figure 6(c) shows the sequence number progression of a sender's transport layer for both PRISM and vanilla-TCP. As shown in the figure, PRISM can achieve 310% more throughput than vanilla TCP in the presence of both loss-rate and delay disparities (from 0s to 40s) thanks to its fast loss-recovery mechanism (see the magnified graph in Figure 6(c)). Furthermore, PRISM yields 208% better performance than TCP in case of a large bandwidth disparity (ranging from 60s to 90s) from its effective scheduling mechanism and ACK re-sequencing mechanism.

8 Performance Evaluation

We also evaluated PRISM via in-depth simulation in diverse environments.We begin with a simulation model and then evaluate PRISM with respect to bandwidth aggregation, packet reordering, and network utilization.

8.1 Simulation Models

We use the *ns-2* [3] for our simulation study. The network topology in Figure 10 is used for this study and consists of Internet, WWAN, and WLAN networks and nodes. The Internet is composed of fixed servers (sender S_i), a proxy, and other hosts ($Host_{S/R}$) for background traffic. The bandwidth between hosts and their edge router is 20 Mbps, and the bandwidth between routers is 10 Mbps.

For WWANs, we use the Universal Mobile Telecommunication System (UMTS) *ns-2* extension [5]. B_i is a base-station node that has support for WWAN links. For WLANs, we use the IEEE 802.11b implementation in *ns-2*, and add NOAH [21] routing protocol to simulate peer-to-peer communication among community members. For each community member, we use an *ns-2* mobile node with extension for supporting multiple wireless interfaces.

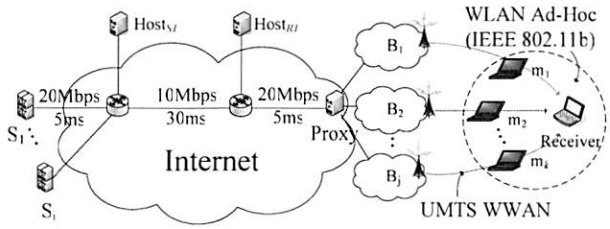


Figure 10: *Simulation network topology.* S_i (server), $Host_{S,R}$ (background traffic generator), a Proxy, B_j (base station), m_k (community members), and R (receiver)

We set up a PRISM flow(s) between a sender(s) and a receiver(s), and measure the end-to-end throughput between them. We implement TCP-PRISM (an extension of TCP SACK) for the sender's transport layer, and place PRISM-IMUX at the proxy. We run an FTP application between the sender and the receiver for 150–500 seconds.

8.2 Achieving Bandwidth Aggregation

We measured aggregated bandwidth gains by PRISM while increasing the number of WWAN links. For a scenario with i links, we randomly choose each link bandwidth between 400 Kbps and 2.4 Mbps. We first run an FTP application between a server (S_1) and a receiver (R_1) for 300 seconds under PRISM, and then run the same experiment without the proxy ('No Proxy'). For better comparison, we also run the same experiment under a weighted-round-robin (WRR) striping agent without an ACK-control mechanism.

PRISM achieves the aggregated bandwidth that reaches the sum of link bandwidths, and its performance scales well with the community size. Figure 7 plots the bandwidth aggregation gain by PRISM and confirms the performance gain and scalability with up to five community members.[4] By contrast, using the WRR striping agent, TCP performance degrades to the one that is worse than a single community member's throughput due to frequent out-of-order packet deliveries. Note that the "Ideal" case is defined as the sum of vanilla-TCP's throughputs achieved in each WWAN link.

8.3 Minimizing Need for Packet Reordering

8.3.1 Bandwidth disparity

We evaluated ADAS's performance in the presence of disparity between WWAN links' bandwidths. We use three community members whose bandwidth difference (say d) increases from 0% to 70%, and measure the achieved aggregate throughput. We initialize the WWAN bandwidth of all members to 1.4 Mbps. Then, we increase one member's bandwidth by d% of 1.4 Mbps and decrease the bandwidth of one of the remaining members by the same percentage. We disable RPC to isolate the performance benefits of ADAS, and run PRISM with other existing scheduling mechanisms as well as ADAS for comparison.

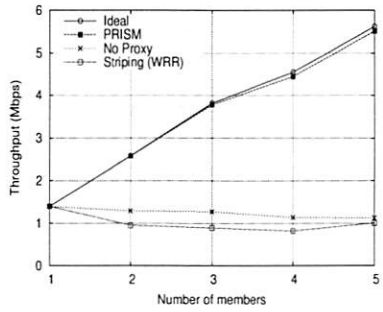


Figure 7: Bandwidth aggregation in the mobile community with different community sizes

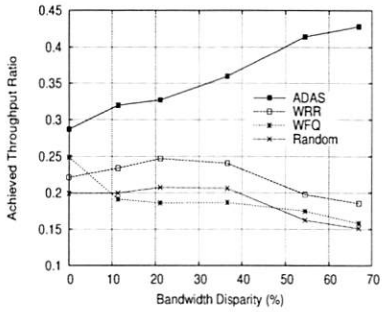


Figure 8: PRISM performance comparison in B/W disparity for different scheduling schemes without RPC.

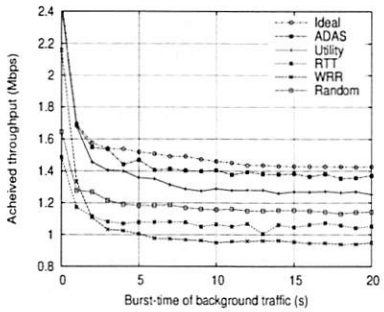


Figure 9: PRISM performance comparison under rate/delay fluctuations for different scheduling schemes.

ADAS reduces out-of-order packet deliveries by sensing bandwidth disparity, and improves links utilization. Figure 8 shows the performance gains on reducing the need for packet reordering under various scheduling mechanisms. The x axis represents the bandwidth disparity, and the y axis is an achieved throughput which is normalized by the ideal total bandwidth of WWAN links. Although the maximum ratio is below the half of ideal bandwidth due to the absence of RPC, the figure shows that the throughput by ADAS improves as the bandwidth disparity increases by selectively using high-bandwidth link bandwidth, meaning that ADAS reduces out-of-order deliveries.

On the other hand, since other scheduling mechanisms such as WRR and RR blindly assign packets to all available links without being aware of bandwidth disparity, their performance is degraded by the use of low-bandwidth links, which causes significant out-of-order deliveries.

8.3.2 Rate/delay fluctuation

We also evaluated ADAS's adaptivity to rate/delay fluctuations by examining end-to-end throughput given dynamic background traffic. Having three community members (whose WWAN bandwidths are 600, 900 and 1200 Kbps, respectively), we run one PRISM flow and two On/Off background traffic (one to the first member's WWAN link with 400 Kbps, and the other to the third with 800 Kbps). We use a burst-time of On/Off traffic as a parameter of rate/delay fluctuation with a Pareto distribution and a fixed idle-time (1s). At this time, we enable RPC functions to show the overall performance improvement.

ADAS adapts to the rate/delay fluctuation of WWAN links and reduces the need for packet reordering. As we will see in Section 8.4.2, reduced out-of-order packet deliveries makes an end-to-end throughput improvement, so we measured the throughput achieved by several scheduling algorithms while increasing rate/delay fluctuations. As shown in Figure 9, ADAS outperforms the other scheduling mechanisms by 12% to 47% in the presence of maximum background traffic.

On the other hand, WRR performs worse than random scheduling in the presence of large fluctuations. $hRTT$-only scheduling exhibits worse performance than the others because the fast but low-bandwidth link limits the overall performance by dropping most of packets. A utility-only scheduling algorithm provides similar performance to ADAS under stable links state. However, as the rate/delay fluctuates more, the U-only scheduling becomes less responsive to short-term fluctuations than ADAS which adapts itself to the fluctuations by using RTT, and thus achieves only 88% of ADAS's throughput.

8.4 Maximizing Network Utilization

8.4.1 Performance gains by RPC

We evaluated the RPC's benefits in network utilization. We use the same setting as in the bandwidth disparity experiment, and for better comparison, we compare three cases: PRISM without RPC, PRISM with only ACK resequencing (partial RPC), and PRISM with full RPC (including loss detection and fast loss recovery).

RPC achieves maximum network utilization by which PRISM can deliver almost ideal aggregated bandwidth. Figure 11 shows the performance gains achieved by RPC. PRISM with the full RPC indeed achieves maximum network utilization even in the presence of large bandwidth disparities. On the other hand, PRISM's performance without RPC shows less than 50% of ideal bandwidth. PRISM with a partial RPC yields, on average, only a 50% performance improvement since it should depend only on timeouts for packet-loss recovery.

8.4.2 Minimizing traffic burstiness

We evaluated the ADAS's contribution in network utilization by measuring the degree of traffic burstiness that depends on the scheduling mechanism. We use four community members (whose WWAN bandwidths are 620, 720, 720, and 860 Kbps), and measure the size of re-sequencing buffer in PRISM-IMUX while running a PRISM flow with ADAS. We run PRISM with WRR for comparison.

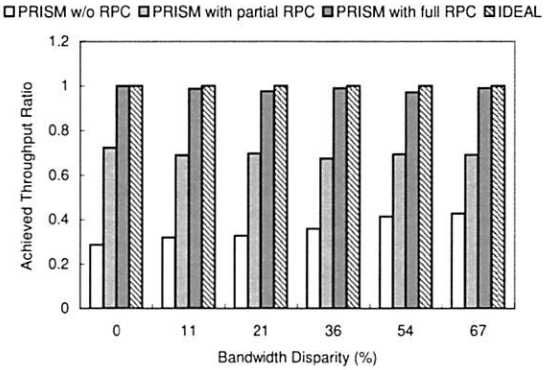


Figure 11: Performance gains by RPC

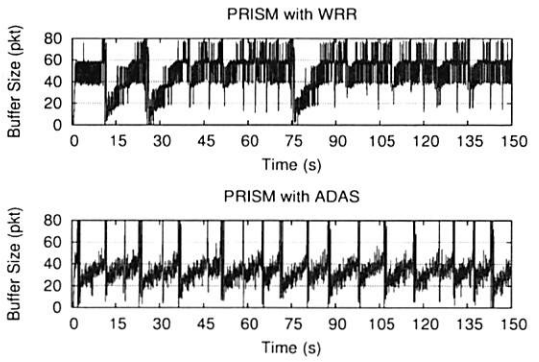


Figure 12: Re-sequencing buffer size progression

ADAS reduces traffic burstiness by minimizing out-of-order deliveries, and thus improves overall network utilization. Figure 12 shows the progression of the re-sequencing buffer size which is defined as the distance between *left* and *highest* of the re-sequencing buffer. The average buffer size required by ADAS in the lower figure is 1.5 times less than that by WRR, meaning that ADAS generates less out-of-order packet deliveries than WRR. Also, the ADAS's smaller buffer size requirement implies a reduced chance for bursty traffic because PRISM-IMUX releases only a small number of stored ACKs to the sender. Our experimental results show that the throughput (2.9 Mbps) for less bursty traffic (scheduled by ADAS) improves up to 16% over the throughput (2.5 Mpbs) for bursty traffic (scheduled by WRR).

9 Related Work

Bandwidth aggregation in multi-homed mobile hosts is considered by several researchers. pTCP [12] and R^2CP [11] are transport-layer approaches to achieving aggregated bandwidth. They make a transport protocol have multiple states so that the transport layer can open multiple connections through multiple interfaces in a *single* mobile host. MOPED [8] is a framework to enable group mobility such that a *single* user's set of personal devices appear as a single mobile entity connected to the Internet.

Packet reordering is a major problem in multi-path routing environments. DSACK [22] in TCP is a detection mechanism of spurious retransmissions on packet reordering based on the information from a receiver via DSACK block. TCP-Door [20] is another scheme for detecting packet reordering in a MANET environment. This approach uses additional information, called *TCP packet sequence number*, to detect out-of-order packets. However, these mechanisms can solve occasional (but not persistent) out-of-order packet arrivals. TCP-PR [6] addresses this problem using a timeout as an indication of packet loss instead of duplicate ACKs, but it may still suffer from false timeouts that result from large RTT fluctuations.

Scheduling packets across multiple links is a well-known problem, and there are three approaches: Round-Robin (RR), fair-queuing, and hybrid. First, the RR scheduling guarantees long-term fairness, and is of low complexity [4]. However, RR inherently causes traffic burstiness which may require a large re-sequencing buffer. Second, the fair queuing attempts to approximate the Generalized Process Sharing (GPS) to achieve fairness (e.g., PGPS, WFQ, WF^2Q). However, these approaches assume that the exact bandwidths of each input and output link are known, which is expensive for resource-limited mobile hosts to obtain. Finally, a hybrid approach (e.g., [16]) removes the complexity of the fair-queuing approach, but it also assumes the known/fixed service rate.

10 Discussion and Conclusion

10.1 Discussion

PRISM can easily support upstream traffic (from a mobile host to a server) by placing PRISM-IMUX at a mobile node in the community. One mobile member in the community can work as the proxy and inverse-multiplex traffic over other community members. It might incur overheads to mobile hosts, but, as shown in Sections 4, 5, and 6, the computational complexity of PRISM increases only on a log-scale, and its spatial complexity is also reasonable (3KB). Most of all, fast transmissions at an aggregate high data rate via members' collaboration contribute to the savings of a base power of mobile hosts. Quantifying this benefits is part of our future work.

We also consider two different security-related issues: (i) what if the packet header is encrypted? and (ii) what if a community member behaves maliciously? Since PRISM exploits TCP information, it is critical for PRISM to extract the header information from each packet. As was done in [18], if we consider the proxy as a trusted party and let it hold the secret key for each connection, then the proxy can extract the header information from encrypted packets. This mechanism also helps prevent members' malicious behaviors from tampering with, or extracting data from, a packet. The other approach to the members' malicious behavior problem is to have a reputation and

punishment system as in [7] to discourage such behaviors.

10.2 Concluding Remarks

In this paper, we first demonstrated the need for a mobile collaborative community: it improves the user-perceived bandwidth as well as the utilization of diverse wireless links. Then, we addressed the challenges in achieving bandwidth aggregation for a TCP connection in the community. Striping a TCP flow over multiple wireless WAN links requires significant scheduling efforts due to heterogeneous and dynamic wireless links, creates the need for frequent packet reordering due to out-of-order packet deliveries, and causes network under-utilization due to the blind reaction of the TCP's congestion control mechanism.

To remedy these problems, we proposed a proxy-based inverse multiplexer, called *PRISM*, that effectively stripes a TCP connection over multiple WWAN links at the proxy's network layer, masking adverse effects of out-of-order packet deliveries by exploiting the transport-layer information from ACKs. PRISM also includes a new congestion control mechanism that helps TCP accurately respond to the heterogeneous network conditions identified by PRISM.

Through experimental evaluation on a testbed and in-depth simulations, PRISM is shown to opportunistically minimize the need for packet reordering, effectively achieve the optimal aggregate bandwidth, and significantly improve wireless links utilization.

Acknowledgement

The authors would like to thank Jack Brassil, Sung-Ju Lee, and Puneet Sharma of HP Laboratories for introducing the concept of mobile community. The work reported in this paper was supported in part by AFOSR under Grant No. F49620-00-1-0327.

References

[1] Netfilter. http://www.netfilter.org.

[2] Nist net. http://snad.ncsl.nist.gov/nistnet.

[3] *ns-2* network simulator. http://www.isi.edu/nsnam/ns.

[4] H. Adiseshu, G. Parulkar, and G. Varghese. A reliable and scalable striping protocol. In *Proceedings of the ACM SigComm*, Stanford, CA, Aug. 1996.

[5] A. Baiocchi and F. Vacirca. End-to-end evaluation of WWW and file transfer performance for UMTS-TDD. In *Proceedings of the IEEE GlobeCom*, Taipei, Nov. 2002.

[6] S. Bohacek, J. P. Hespanh, J. Lee, C. Lim, and K. Obraczka. TCP-PR: TCP for persistent packet reordering. In *Proceedings of the 23rd ICDCS*, Rhode Island, May 2003.

[7] S. Buchegger and J.-Y. L. Boudec. Performance analysis of the CONFIDANT protocol:cooperation of nodes. In *Proceedings of the ACM MobiHoc*, Lausanne, Switzerland, June 2002.

[8] C. Carter and R. Kravets. User device cooperating to support resource aggregation. In *Proceedings of the 4th IEEE WMCSA*, Callicoon, NY, June 2002.

[9] J. Duncanson. Inverse multiplexing. *IEEE Communications Magazine*, 32(4), Apr. 1994.

[10] D. Farinacci, S. Hanks, D. Meyer, and P. Traina. Generic routing encapsulation (GRE). Internet Request for Comments 2784 (rfc2784.txt), Mar. 2000.

[11] H. Hsieh, K. Kim, Y. Zhu, and R. Sivakumar. A reciver-centric transport protocol for mobile hosts with heterogeneous wireless interfaces. In *Proceedings of the ACM MobiCom*, San Diego, CA, Sept. 2003.

[12] H. Hsieh and R. Sivakumar. A transport layer approach for achieving aggregate bandwidths on multi-homed mobile hosts. In *Proceedings of the ACM MobiCom*, Atlanta, GA, Sept. 2002.

[13] L. Magalhaes and R. Kravets. MMTP:multimedia multiplexing transport protocol. In *Proceedings of SigComm-LA*, San Jose, Costa Rica, Apr. 2001.

[14] S. Marti, T. Giuli, K. Lai, and M. Baker. Mitigating routing misbehavior in mobile ad hoc networks. In *Proceedings of the ACM MobiCom*, Boston, MA, Aug. 2000.

[15] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP selective acknowledgement options. Internet Request for Comments 2018 (rfc2018.txt), Oct. 1996.

[16] S. Ramabhadran and J. Pasquale. Stratified round robin: A low complexity packet scheduler with bandwidth fairness and bounded delay. In *Proceedings of the ACM SigComm*, Karlsruhe, Germany, Aug. 2003.

[17] P. Rodriguez, R. Chakravorty, J. Chesterfield, and I. Pratt. Mar: A commuter router infrastructure for the mobile internet. In *Proceedings of the ACM MobiSys*, Boston, MA, June 2004.

[18] N. B. Salem, L. Buttyan, J.-P. Hubaux, and M. Jakobsson. A charging and rewarding scheme for packet forwarding in multi-hop cellular networks. In *Proceedings of the IEEE/ACM MobiHoc*, Annapolis, MD, June 2003.

[19] J. Veizades, E. Guttman, C. Perkins, and S. Kaplan. Service location protocol. Internet Request for Comments 2165 (rfc2165.txt), June 1997.

[20] F. Wang and Y. Zhang. Improving TCP performance over mobile ad-hoc networks with out-of-order detection and response. In *Proceedings of the ACM MobiHoc*, Lausanne, Switzerland, June 2002.

[21] J. Widmer. Network simulations for a mobile network architecture for vehicles. http://www.icsi.berkeley.edu/widmer/mnav/ns-extension.

[22] M. Zhang, B. Karp, and S. Floyd. Improving TCP's performance under reordering with DSACK. Technical report, International Computer Science Institute, Technical Report ICSI TR-02-006, July 2002.

[23] S. Zhong, J. Chen, and Y. Yang. Sprite: A simple, cheat-proof, credit-based system for mobile ad-hoc networks. In *Proceedings of the IEEE InfoCom*, San Francisco, CA, Apr. 2003.

Notes

[1] We assume that the community is formed in such a way that its members have mutually exclusive frequency channels to make bandwidth aggregation practical if they subscribe to the same ISP.

[2] It could be the second block when a DSACK option is used.

[3] We use the on-line resource of [17].

[4] Note that we limit the maximum community size to 5 since the IEEE 802.11b provides up to 6 Mbps in terms of end-to-end throughput.

Horde: Separating Network Striping Policy from Mechanism

Asfandyar Qureshi and John Guttag
MIT Computer Science and AI Laboratory
{asfand, guttag}@csail.mit.edu

Abstract

Inverse multiplexing, or network striping, allows the construction of a high-bandwidth virtual channel from a collection of multiple low-bandwidth network channels. Striping systems usually employ an immutable packet scheduling policy and allow applications to be oblivious of the way in which packets are routed to specific network channels. Though this is appropriate for many applications, other applications can benefit from an approach that explicitly involves the application in the dynamic determination of the striping policy.

Horde is middleware that facilitates flexible striping in wireless environments for a diverse range of applications. Horde separates the striping policy from routing and scheduling. It allows applications to specify network quality-of-service objectives that the striping mechanism attempts to satisfy. Horde can be used by a set of application data streams, each with its own quality-of-service policy, to flexibly stripe data over a highly heterogeneous set of dynamically varying wireless network channels.

We present the Horde architecture, describe an early implementation, and examine how different policies can be used to modulate the quality-of-service observed across different independent data streams.

1 Introduction

Horde is networking middleware that provides a simple and robust way for multi-stream applications to communicate over multiple channels with widely varying latency and bandwidth. The key problems it addresses are:

- providing applications with a way to influence the scheduling of packets over channels, without building into the applications knowledge about channel availability or characteristics, and
- providing a mechanism that uses this information to derive appropriate packet transmission schedules for these time-varying channels.

Our work on Horde was motivated by our inability to find an existing solution to support the development of a system on which we were working. As part of a telemedicine project, we wanted to transmit real-time uni-directional video (on the order of 300kbps), bi-directional audio, and uni-directional physiological data streams (EKG, blood pressure, etc) from a moving ambulance. By relaying real-time telemetry and video from ambulances, we hope to provide EMS teams with expert opinions on complex trauma injuries, and to aid the in-hospital teams in better preparing themselves for incoming patients. Our telemedicine system must be economically viable to build, deploy, and operate. We therefore will leverage existing communications infrastructure, instead of building our own network infrastructure.

In most urban areas, there are a large number of public carrier wireless channels providing mobile connectivity to the Internet (e.g., GPRS and CDMA). The upstream bandwidth offered by these Wireless Wide Area Network (WWAN) channels is typically rather limited and each channel provides little in the way of network Quality-of-Service (QoS) guarantees.

These issues led us to consider using inverse multiplexing, or network striping, to aggregate several of these WWAN channels to construct a virtual channel. Network striping takes data from the larger source channel and sends it in some order over the smaller channels, possibly reassembling the data in the correct order at the other end before passing it to the application. By taking advantage of service provider diversity, overlapping coverage, and network technology diversity, we can use striping to provide an application with the illusion of a reliable stable high-bandwidth channel.

	Mean (μ)	Sdev (σ)
GPRS upload bandwidth (stationary)	$25kbps$	1
(moving)	$19kbps$	5
CDMA upload bandwidth (stationary)	$130kbps$	5
(moving)	$120kbps$	22
GPRS small packet RTT (stationary)	$560ms$	100
(moving)	$760ms$	460
CDMA small packet RTT (stationary)	$460ms$	90
(moving)	$470ms$	120
768-byte packet RTT (CDMA)	$810ms$	
(GPRS)	$920ms$	

Figure 1: Summary of WWAN QoS characteristics.

1.1 WWAN Striping Challenges

A great deal of work has been done on network striping [1, 8, 9, 12, 15, 18]. Most of this work is aimed at providing improved scheduling algorithms under the assumption that the underlying links are relatively stable and homogeneous, and that application streams are similar (e.g., TCP). If these assumptions hold, there is little reason to give applications control over how the striping is done, and allowing applications to be oblivious to the fact that striping is taking place is advantageous.

However, these assumptions of homogeneity and stability are unrealistic for the WWAN channels we are using [6, 14, 15]. The bandwidth/latency characteristics of the available WWAN channels can vary by as much as an order of magnitude. In addition to heterogeneity among channels, we also expect there to be a high degree of temporal variation in QoS on each WWAN channel. QoS varies in time, partly because of vehicular motion and partly because of competition with other users on the WWAN. Spatial variation of the QoS depends on the carrier's placement of cell-towers relative to the terminal.

Our experiments with existing WWANs (GSM/GPRS and CDMA2000 1xRTT) in the Boston area provide evidence of heterogeneity and high QoS variability [14]. Figure 1 summarizes our experimental observations.

Since the WWANs are neither stable nor homogeneous, the manner in which the middleware decides to schedule the transmission of application packets can have a large influence on data stream latencies, bandwidth, and loss rates. Furthermore, the data streams in our telemedicine system are heterogeneous with respect to which aspects of the network service they are sensitive to: some streams care about latency (e.g., video streams), some not (e.g., bulk-data transfers); some care about loss more than others (e.g., audio); and some care more about jitter than they do about latency (e.g., non-interactive video). Therefore we want to give the application some control over how striping is done.

1.2 Horde's Approach

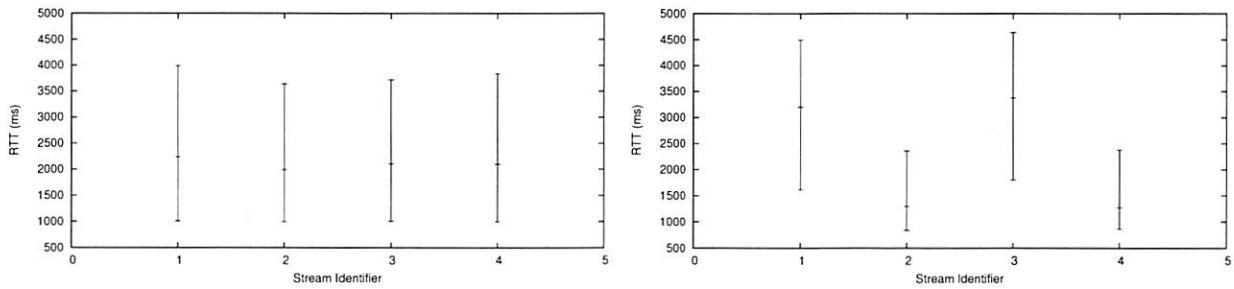
Horde separates the striping *mechanism* from the striping *policy*, the latter being specified by the application in an abstract manner. The key technical challenge in Horde is giving the application control over certain aspects of the data striping operation (e.g., an application may want urgent data to be sent over low latency channels or critical data over high reliability channels) while at the same time shielding the application from low-level details. Horde does this by exporting a set of flexible abstractions to the application, in effect replacing the application's network stack.

In addition to aggregating bandwidth, Horde allows an application to modulate network QoS for its streams. Horde allows an application to express its policy goals as succinct network QoS *objectives*. Each objective says something, relatively simple, about the *utility* an application gains from some aspect of network QoS on a stream. Objectives can take into account such things as expected latencies, observed loss-rates, and expected loss correlations. Using the set of expressed application objectives, Horde attempts to schedule packets at the sender so as to maximize the expected utility derived by the application from the resulting sequence of packet receptions.

Allowing an application to actively influence the striping operation can be beneficial. By allowing the application to express its desired goals, Horde can arrive at efficient transmission schedules that provide high utility.

Consider, for example, a simplified version of our telemedicine application. There are four data streams: EKG physiological data (stream 1), video (stream 2), additional physiological data (stream 3), and audio (stream 4). Figure 2a shows how packet latencies were distributed on these streams when striping over three WWAN channels, using a packet scheduler that did not distinguish between the data streams. All streams received roughly the same QoS. However, two of these streams (video and audio) are latency sensitive. An objective can be used that expresses that smaller packet latencies on the video and audio streams give the application more utility than larger latencies. Figure 2b shows that, with this single objective, Horde was able to provide lower-latency for the latency sensitive streams.

The telemedicine system contains more complex examples where QoS modulation is advantageous. An encoded video stream may contain both reference frames (I-frames) and delta frames (P-frames). In order for P-frames to be decoded properly at the receiver, the I-frames they depend on must have been successfully transmitted across the network. An application therefore derives more utility from an I-frame than it does from



(a) Randomized Round Robin

(b) Objective Driven Scheduling

Figure 2: Packet round-trip-time distributions with different schedulers, striping over three channels. The two schedulers were run over the same packet traces, collected from three existing WWAN channels (one CDMA and two GPRS channels). The graphs show the median and the upper and lower quartile packet latencies for the streams.

a P-frame. This suggests that I-frames should be sent over more reliable channels. If two I-frames from different video layers contain similar information, it may make sense to ensure that they are sent so that I-frame losses are uncorrelated. Additionally, during playback the audio must be kept synchronized with the video. In this case, an application may value receiving an audio packet at roughly the same time as the related video packets.

The Horde middleware is designed to ease the development of applications that care about network QoS and need to use WWAN striping. Application-specific striping code has been used in the past. Many multi-path video streaming applications [3, 5, 16] exemplify this approach. In these systems, striping code is intertwined with the application logic. In contrast, Horde allows one to build a multi-channel video streaming application that cleanly separates the network striping and network channel management operations from the application logic.

Horde is not meant to be general networking middleware. Existing applications would have to be rewritten to take advantage of Horde's QoS modulation framework. Further, as long as most application data can be sent down a single, stable link, using Horde is overkill. More generally, in situations where one is dealing with a fixed set of relatively homogeneous and stable channels, other techniques [1, 10, 18] may be more appropriate.

Horde is most useful when dealing with:

Heterogeneous Data Streams When different streams gain value from different aspects of network performance, trade-offs can be made when allocating network resources among those streams.

Heterogeneous/Time-varying Network Channels With such channels, the scheduler has an opportunity to significantly modulate QoS. Modulation can be more accurate when channel characteristics are predictable in (at least) the short term.

Bandwidth-Limited Application Applications that want to send more data than individual physical channels can support justify both the network striping operation and the additional processing cost of Horde's QoS modulation framework.

Single Application Our model is that of a single application with multiple streams. While the application may be split across multiple processes, we assume a consistent system-wide policy in this paper. We therefore ignore the effects of adversarial behaviour.

1.3 Paper Organization

We have introduced the problem domain and outlined our approach and motivations above. Section 2 presents an overview of the Horde architecture. Section 3 discusses our approach to separating application policy from the striping mechanism. Section 4 presents some experimental results. Section 5 describes related research. Finally, section 6 presents a summary and conclusion.

2 Horde Architecture

Horde provides to an application the ability to:

- Stripe data streams over a dynamically varying set of heterogeneous network channels;
- Abstractly define striping policy;
- Per-network-channel congestion control;
- Explicit flow control feedback for each stream.

This section presents a high-level overview of Horde's application interface and internal structure. We also discuss selected aspects of Horde. We defer discussion of the scheduler and policy interface to section 3.

2.1 Overview

Application Interface

Horde uses a connection-based model. The applications on the source and destination nodes must negotiate a connection, or *stream*, before they start sending data to each other. An application can register event callbacks for each data stream it creates. When sending/receiving data on a stream, applications communicate with Horde at the granularity of ADU's. Our design draws from previous arguments for application level framing [7, 2].

Horde manages stream flow-control. Streams request and receive bandwidth allocations. Additionally, for each stream, Horde sends *throttle* events to the application to notify it about changes in allocated bandwidth.

If the application is sensitive to some QoS aspects on any stream, it can inject *objectives* into the middleware to modulate the network QoS for those streams.

Internal Structure

Internally, Horde is divided into three layers (figure 3). The lowest layer presents an abstract view of the network channels to the higher layers of the middleware. This layer deals directly with the network channels, handling packet transmissions, congestion control, and probes. The middle layer, is composed of the inverse multiplexer and the bandwidth allocator. The highest layer interfaces with application code.

Applications use the interface provided by the highest layer to inject and remove ADU's and policies. Horde also delivers ADU's and invokes stream event handler callbacks using this interface. Our implementation of Horde is in user-space. In our implementation, the highest layer provides a simple IPC interface to Horde[1].

In the middle layer, the outgoing packet scheduler decides how to schedule ADU's from the unsent ADU pool over the available channels. Incoming packets are delivered to the relevant data streams by the `pReceiver` module. The bandwidth allocator, `bwAllocator`, divides up the bandwidth, provided by the channel managers, among the data streams.

In the lowest layer, the channel managers deal directly with the network channels. The channel pool manager monitors network connectivity, making sure there is an appropriate manager for each active channel.

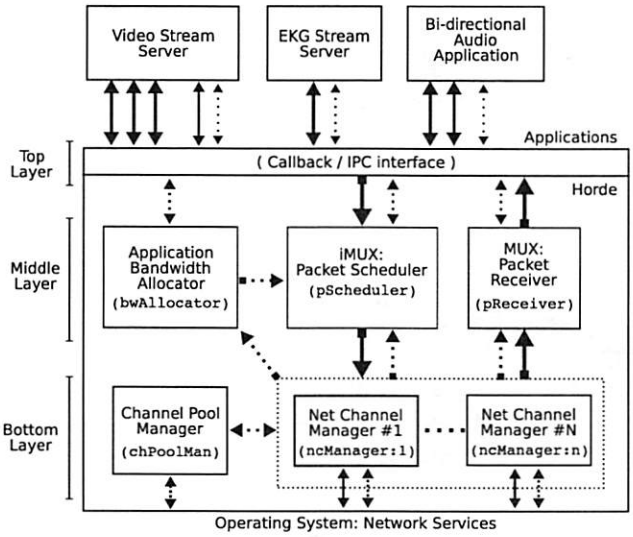


Figure 3: Modular breakdown of Horde. Solid arrows show data flow; dashed arrows represent control signals.

2.2 Network Channel Managers

Horde uses a set of Network Channel Manager modules (the `ncManagers`) to manage the available network channels. Each `ncManager` handles network I/O, maintains a predictive model for the QoS on the channel, and also performs congestion control on the channel.

The generic `ncManager` interface allows different channel models and congestion control schemes to coexist without complicating the packet scheduler. The behaviour of the last hop wireless link can often dominate in determining how QoS varies and how well a congestion control scheme works [4, 14, 17]. Therefore, there may be multiple implementations of the `ncManager` interface, each optimized for a different type of network (e.g., 802.11, CDMA2000, GPRS).

2.3 Network Congestion Control

Congestion control in a striping system should be implemented below the striping layer, independently for each channel. When data is being striped to a single destination, since there are multiple independent channels, there are multiple independent congestion domains. Thus each channel manager in Horde runs an independent congestion control session[2].

We do not believe that having a single congestion control session straddling channels—as previous striping systems have often done—is the right approach. An application, above the striping layer, does not have enough information to implement efficient congestion control,

unless the application can perfectly simulate the scheduling algorithm. For instance, approaches based on loss as an indicator for network congestion may either be inefficient (e.g., if some of the underlying channels provide explicit congestion notification ECN feedback) or just plain wrong (e.g., a loss on one channel may only indicate that the application should send less data down that particular channel, not that it should decrease it's overall sending rate, since other channels may have spare bandwidth). Older systems [1, 18] realized the need for some form of minimal per-channel congestion control below the striping layer, even though TCP congestion control was active above the striping layer.

2.4 Stream Flow Control

Bandwidth allocation is separated from QoS modulation in Horde. Different scheduling strategies with a bandwidth allocation can modulate QoS in different ways. For example, n bytes of a stream's data can be sent on a fast channel or spread over fast and slow channels.

Conceptually, Horde's interface separates the quantity of service a stream receives (the number of slots) from the quality of that service (the latency/loss/etc of those slots). Of course, this is not always a clean separation: a QoS with high loss can lead to low goodput.

In Horde, the QoS modulation mechanism is constrained by the bandwidth allocation mechanism. When producing transmission schedules, the amount of data sent is restricted based on the stream allocations.

Every active Horde stream is allocated some fraction of the available bandwidth. For each stream, an application either specifies the maximum rate at which it wants to send data, or marks that stream as being driven by an *elastic* traffic source. Horde in return informs the application the rate at which it can send on that stream. The present implementation of Horde uses a simple adaptive min-max fairness policy to allocate available bandwidth among streams. If a stream sends more than its allocated rate, ADU's are dropped in an unspecified order inside the middleware, because of sender-side buffer overruns.

As available bandwidth changes, `throttle` events—reflecting the new bandwidth allocations—are generated by Horde and delivered to each stream manager callback. Simple streams can ignore these events, always sending at their maximum rates, letting the middleware decide which ADU's to drop. Conversely, adaptive streams can use the information conveyed by the events to guide changes to their behaviour. Some streams may respond to these events by ramping their sending rates up or down. Others may change their behaviour in more complex ways, for example: down-sampling the stream; changing stream encodings; omitting less important ADU's, etc.

Enforced bandwidth allocations and explicit flow-control feedback are important in providing graceful degradation when independent data streams exist. Available bandwidth will often be outstripped by total demand. Since callbacks managing different streams can be independent of each other, a mediator is needed inside the middleware to divide the bandwidth *fairly*. This mediator transforms changes in total bandwidth into some set of changes in the bandwidth of each stream.

2.5 Transmission Slots

Horde uses the notion of transmission slots, in the form of `txSlot` objects, to provide an abstract interface between the channel manager modules and the packet scheduler. A `txSlot` is an abstract representation for the capability to transmit some data on a specific network channel at a given time, along with the expected QoS for that transmission. The fields of the `txSlot` abstraction are shown in figure 4. As we discuss later in section 3, the scheduler makes its decisions based solely on the information in the `txSlot` objects. This significantly simplifies the scheduling problem.

Transmission Capabilities Each `txSlot` object represents a *capability* that the `ncManager` grants to the scheduler, allowing the transmission of data on that channel. The generation of `txSlot`'s is governed by the `ncManager`'s congestion control logic. In a scheduling cycle, the scheduler acquires slots from all active channel managers, maps ADU fragments to those `txSlot`'s, and passes that mapping back to the managers, resulting in data being transmitted.

Latency/Loss Expectations For every `txSlot`, the parent `ncManager` uses a dynamically updated probabilistic model of the channel's behaviour to derive the expectations for that slot. A simple `ncManager` implementation could use a weighted moving average of previously seen round-trip-times to determine the expected latency and an average ($\frac{total\ delivered}{total\ lost}$) for loss[3].

Correlated Loss When two slots suffer losses, we say that the losses in those slots are *correlated*. There are two important types of correlated losses: burst losses on a single channel and correlated losses on different channels. We have observed that individual WWAN channels exhibited bursty losses. Correlated losses on different

Field	Description	Range
`channelID`	Parent network channel for this slot.	
`sequence`	The sequence number for this slot on its parent channel.	
`lossProbability`	The estimated loss probability for a packet transmitted in this slot.	$loss \in [0, 1]$
`expectedRTT`	Expected time between the transmission of data in this slot and the reception of an acknowledgment.	$milliseconds \geq 0$
`lossCorrelation(other)`	Compares two transmission slots to see if packet losses in the two slots are expected to correlate. This is an estimate for $P(both\ lost \mid either\ one\ lost)$.	$correl \in [0, 1]$
`cost(x)`	Cost of transmitting x bytes in this slot.	$cost(x) \geq 0$
`maximumSize`	The maximum number of bytes that can be transmitted in this slot.	$size > 0$

Figure 4: The main components of Horde's transmission slot (`txSlot`) abstraction.

channels also occur, because of signal fading or external factors that cause cross-channel correlation (e.g., when two channels have antennas on a tower that is occluded).

Striping presents multiple opportunities to reduce correlated losses. ADU's can be sent down different channels, provided these channels exhibit uncorrelated losses. ADU's can also be spaced out temporally on the same channel, interleaving multiple data streams to maximize throughput and reduce correlated losses on the streams.

The `txSlot` interface provides a loss correlation metric to help the scheduler make judgments about correlated losses. Reasoning about loss correlations can be important in some application domains (e.g., streaming multiple description video). As figure 4 indicates, two `txSlot` objects can be compared to see if a loss in one slot is expected to correlate with a loss in the other slot. The result of this comparison is a conditional probability for the event that both slots will experience losses if either of them experiences a loss.

Transmission Costs There are often monetary costs associated with the transmission of data on a channel. A `ncManager` can maintain its own cost model, configured with provider-specific information. For instance, for a WWAN account with a data transmission quota, the cost for each slot above the quota is higher than for earlier slots. Since the cost model for each parent channel is accessible from the appropriate `txSlot`'s, cost models can be used in policy decisions by the scheduler.

Phantom Transmission Slots A channel manager can also be asked to *look-ahead* into the near future (e.g. the next second) to estimate how many more slots are likely to become available on that channel. Managers can provide *phantom* `txSlot` objects, each tagged with a confidence level, a measure of how sure the channel manager is of its prediction. Phantom slots are discussed more fully in section 3.3.

3 Policy and Scheduling

At the core of any striping system lies a packet scheduler that decides how to transmit packets from data streams on the multiple network channels. The packet scheduler does not determine the data-rate for a stream. In Horde, the bandwidth allocator's policy, in conjunction with the congestion control algorithms running within the channel managers, ultimately limit stream data-rates. The scheduler must then decide which transmission slot carries which ADU.

The scheduler implements a *policy*. The scheduler transforms its policy into transmission schedules based on the offered load and using information from the `txSlot`'s, about the present and near-future behaviour of those channels. For example, a scheduler could have the policy of minimizing receiver-side ADU reordering in a stream. Such a scheduler may try to minimize reordering as it stripes the data, but—with imperfect predictions about the future—it may not be able to do so.

Most contemporary striping systems use a *static* scheduler policy [1, 10, 18] that cannot be modulated by applications. Often, the policy in these systems is inseparable from the scheduling mechanism itself.

With a heterogeneous and/or dynamically unstable set of channels, the transmission slots can vary considerably in terms of expected loss and latency characteristics, and so the manner in which the scheduler decides to transmit application packets can be crucial in determining the network QoS for a data stream. For example, a scheduler could consistently assign slots with low expected latencies to stream x, following a policy of minimizing the average latency on that stream.

In Horde, the striping policy is separated from the striping mechanism. Horde allows an application to define a time-varying striping policy at run-time, providing a generalized mechanism within the scheduler to facilitate many different policies. An Application expresses its policy goals as modular network QoS objectives, and these objectives drive the scheduler towards transmission

schedules valued highly by that application.

3.1 Application Utility

During a period of network service in which an application actively sends ADU's, it obtains some utility from the consumption of its ADU's at another host. We refer to this abstract utility as the application's *utility function* and represent it as a numerical function—in the same way microeconomics textbooks choose to model the utility consumers assign to goods they consume [13].

The application's utility function represents *net* utility. *Gross* utility refers to the total value derived by the application. Net utility is the difference between the gross utility and the *cost* of the network service. *Expected* net utility can be obtained by using latency and loss expectations as inputs to the utility function.

Utility is directly related to the number, identity and delivery time of the delivered ADU fragments. However, since the sender doesn't know the delivery times, we work with network round-trip-times, derived from the `ack` arrival sequence. Factors such as local queuing time, parent stream identity and ADU type (e.g. I-frame) can also impact utility. The ADU header fields provide this information.

The idealized utility function can be written as:

$$utility_{app}(\tau) \sim f_{app}(history_{\{tx\}}(\tau),\ history_{\{rx\}}(\tau))$$

Where, over a period τ: packet transmission history ($history_{tx}$) is a set of triples of the form $(adu, fragment, txslot)$; the `ack` reception history ($history_{rx}$) is a set of $(adu, fragment, time)$ triples; and f_{app} is an arbitrary application defined function.

$utility_{app}$ defines a partial order on possible transmission schedules, which can be used by the scheduler to rank its scheduling choices. $utility_{stream}$ (used below) is an analogous utility function for a stream.

Of course, the scheduler does not have perfect information about the future. Therefore, we work with the expected utility, using expectations from `txSlot`'s.

3.2 'Optimal' Scheduling

We define an *optimal* scheduler as one that picks the transmission schedules most valued by the application, over some time period, given the state of the network during that period and expectations about the future. These expectations are provided by Horde's channel managers.

For a period τ, such a scheduler has complete information about some things (which ADU's have been sent; which `ack`'s have been received) and expectations about others (latency and loss expectations for ADU's that have been sent but not yet `ack`'d; expectations from unused `txSlot`'s; and expectations from phantom `txSlot`'s).

Over τ, an optimal scheduler would find a schedule that maximizes the sum of all streams' utility functions, using the provided expectations and constrained by the stream bandwidth allocations[4]:

$$maximize\left[\sum_{\forall stream}\left(utility_{stream}(\tau)\right)\right]$$

As posed above, the optimal scheduling problem is a computationally impractical, since scheduling decisions typically need to be made frequently. Consequently, our schedulers only attempt to find *high utility* transmission schedules instead of optimal schedules.

3.3 Objective Driven Scheduling

By providing a specialized scheduler and a sufficiently abstract policy expression interface, Horde allows applications to drive the striping operation. An important design issue was deciding how an application expresses its policy goals and how the Horde scheduler translates these goals into transmission schedules. Horde provides a specification language that allows the expression of application goals as succinct network-QoS *objectives*.

The modularity provided by the use of objectives is intended to simplify application development. For those applications at which we have looked, the objectives tend to be relatively simple, e.g., favour lower loss `txSlot`'s for certain types of ADU's (e.g., video I-frames), favour lower latency slots for some streams, and avoid correlated losses for certain sets of ADU's.

Objectives

Objectives represent the modular decomposition of an application's utility function. The set of expressed objectives drives the packet scheduler towards schedules that provide high utility to applications. Objectives can be injected and removed dynamically, supporting the specification of time-varying utility functions.

An objective defines a QoS goal and describes how the achievement of that goal adds to, or subtracts from, overall application utility.

A goal may be expressed in terms of a set of ADU's (e.g., a stream) or, more generally, in terms of a set of ADU sets (e.g., the sets of I-frame ADU's and P-frame ADU's). Objectives will usually be concerned with the policy of a single data stream but it is sometimes useful to define objectives that straddle multiple streams. For

instance, an objective for multi-description video streaming could express the application goal of minimizing correlated frame losses on the different video data streams.

In Horde, objectives describe value relationships between ADU's and `txSlot`'s. Objectives are independent of the number of channels being used to stripe the data. To simplify this discussion, we assume ADU's are never fragmented. A transmission schedule is then just a mapping of ADU's to `txSlot`'s. Each objective says something about how much an application values the assignment of a particular kind of `txSlot` to a particular kind of ADU.

In determining the value of an assignment, two things are important: the nature of the ADU and the nature of the `txSlot`. ADU's can be differentiated based on their header fields. Some of these are managed by Horde (e.g., `stream_id`) while others are application-specific annotations (e.g., `frame_type`). Slots can be differentiated based on the fields shown in figure 4.

More abstractly, the characteristics of a transmission slot can be seen as a collection of three random variables: latency (real), loss (0 or 1) and cost (real). The expectations (μ) and variances (σ^2) of these variables can be used in evaluating slot assignments.

Goals can be expressed over sets of ADU's and so may refer to the expectation or variance of the loss/latency/cost of the slots assigned to that set. With this view, correlated loss is not a special QoS metric, it can be viewed as the expected loss over a set of ADU's.

When an application does not specify an objective for some QoS aspect of a stream, it is implicitly assumed that the application does not care about that aspect of QoS on that stream. A stream with no associated objectives is likely to be assigned the transmission slots left over after existing objectives have claimed the good slots for their own streams. However, bandwidth allocations are enforced before schedules are constructed, so every stream always receives its fair share of slots[5].

3.4 Objective Specification Language

This section outlines a language whose purpose is to provide a flexible interface in which application objectives can be expressed and evaluated. This language represents a fairly straightforward formalization of the abstract notion of objectives we presented earlier.

We have experimented with other possible interfaces. Our initial interface allowed applications to simply flag streams as being low-latency, low-loss, etc. This interface represents a short-hand, using a small number of objective *idioms*. It proved insufficiently flexible. We also

```
objective {
  context {
     adu:foo { (stream_id == "video1") &&
               (frame_type == "I")  }
     adu:bar { (stream_id == "video1") &&
               (frame_type != "I")  }
  }
  goal { prob(foo::lost?)
         < prob(bar::lost?) }
  utility { foo { 100 } }
}
```

Figure 5: An objective expressing the policy that, for stream `video1`, `txSlot`'s carrying I-frames should have lower loss probabilities than slots for other frames.

```
objective {
   context {
     stream:foo { stream_id == "audio1" }
   }
   goal { foo::latency_ave < 1000 }
   utility { foo { 100 } }
}
```

Figure 6: An objective expressing the policy that the average latency on a stream should be less than one second.

tried allowing applications to export black-box callback functions that perform the $utility_{stream}$ calculation. In our experience, the callback approach made it harder to build an accurate scheduler, because the scheduler had very limited information about the objectives.

The language discussed here seems to provide an adequate level of abstraction, flexibility and programmability. Using literals, ADU's, streams, header fields, and latency and loss probability distributions as basic units, the language allows the construction of complex constraints.

The language is type-safe. Each expression has a well-defined type and there are specific rules governing how expressions can be composed and what the type of the composition will be. The `numeric` type in the language definition only supports integers. Probabilities are represented as percentage values ranging from 0 to 100.

An `objective` definition consists of three sections: a `context` (variable bindings); a `goal` (a predicate); and a `utility`. Figures 5, 6 and 7 show examples. When the `goal` predicate is `true`, the interpreter uses the `utility` section to determine how that `goal`'s fulfillment has affected the utility derived from the associated schedule. Active `objectives` are evaluated by the scheduler in an unspecified order.

```
objective {
  context {
     adu:x { (stream_id == "video1") ||
             (stream_id == "audio1")     }
  }
  goal { true }
  utility {
   x { 5 * (1000 - expected(x::latency)) }
  }
}
```

Figure 7: An objective expressing that the utility derived by an application from certain ADU's depends linearly on how close the round-trip-time is to one second.

Context An `objective`'s `context` section specifies a mapping between the `goal` and a set of ADU's that can be used to achieve that `goal`. Each `context` defines a set of filters on all possible ADU's. For every `ADU` or `stream` variable used in the `goal` or `utility` sections, the `objective`'s `context` contains a filter predicate, expressed in terms of ADU header fields or stream identifiers. Fields like `stream_id` are predefined. Applications can arbitrarily define other fields to selectively activate `objectives` for marked ADU's.

The example objectives in figures 5 and 6 show different types of `contexts`. Figure 6's objective applies to stream `7`. Figure 5 binds two variables: `foo` is always bound to an `I`-frame ADU from stream `17`; and `bar` to a non-`I`-frame ADU from stream `17`. `frame_type` is an example of a field being used to integrate application-specific ADU annotations into Horde's scheduler.

Goal The `goal` section specifies a predicate that can be evaluated on some schedule. `goal` predicates can be reasonably involved: simple probability, boolean and numerical expressions can be progressively composed to produce a boolean function. For example, figure 5's `goal` is `true` whenever an `I`-frame ADU can be sent in a slot that has a lower loss probability than a slot used to send a non-`I`-frame ADU.

Utility The `utility` section specifies how application utility is affected when a `goal` has been met. The `utility` section contains a numeric valued expression for each `ADU` or `stream` variable whose utility is affected. Negative utilities bias the scheduler against schedules with the property specified in the related `goal`; positive utilities promote such schedules. The base utility of transmitting an ADU is zero.

Complex numeric `utility` expressions are possible. Figures 5 and 6 show examples that add a constant utility when their goal is met, but figure 7 uses a more involved `utility` section in which the utility gained from transmitting an ADU varies linearly with expected latency.

```
tx_schedule random_walk_scheduler() {
  // collect all slots w/ look-ahead
  slots = collect_slots(LOOKAHEAD_MSECS);
  // collect as many adus as slots
  adus = collect_adus(slots.size());

  // do a random walk
  int max_util = MIN_INTEGER;
  tx_schedule best_schedule = NULL;
  for (int i = 0; i < WALK_LENGTH; i++) {
    // get a random schedule
    tx_schedule sched =
     create_random_schedule(slots, adus);

    // evaluate all objectives over
    // this schedule and get utility
    int util = evaluate_objectives(sched);

    // is this best schedule?
    if (util > max_util) {
      max_util = util;
      best_schedule = sched;
    }
  }
  return best_schedule;
}
```

Figure 8: Pseudo-C++ for `random-walk` scheduler.

The language does not assign semantic meaning to numeric utility, other than the notion that higher utility is better than lower utility. Consequently, the impact of the constants `100` (figures 5 and 6) and `5` (figure 7) can only be gauged by looking at the constants in other active objectives. This represents a weakness in the language.

3.5 Scheduler Implementation

In constructing the scheduler described here, we have consciously attempted to make it as simple as we could. Our goal for this implementation was to show that even very simple scheduling algorithms can provide enough benefits to justify the Horde QoS modulation framework.

In each scheduling cycle, the `random-walk` scheduler uses a random bounded-search of the transmission schedule space to find what looks like a good schedule, given the set of active objectives. The scheduler creates k random transmission schedules, evaluates all objectives over each of these k schedules and picks the schedule with the highest aggregate utility. In the absence of any objectives, the `random-walk` scheduler works like a

```
tx_schedule
 create_random_schedule(slots, adus) {
  tx_schedule random;

  // randomly reorder
  random_shuffle(slots);
  random_shuffle(adus);

  // produce schedule:
  for (int i = 0; i < adus.size(); i++)
   random.assign_slot(slots[i], adus[i]);

  return random;
}
```

Figure 9: Pseudo-C++ for random schedule constructor.

randomized-round-robin scheduler: any valid mapping of slots to ADU's is equally likely.

A scheduling cycle runs every T milliseconds. During each cycle, the scheduler acquires from each channel manager the currently available `txSlot`'s and the phantom `txSlot`'s for N milliseconds into the future. Based on the number of slots, ADU's are collected from the data streams. The constraints imposed by the bandwidth-allocator govern how ADU's can be collected from each stream. Streams are treated as a FIFO queues.

Figures 8 and 9 describe the scheduler in pseudo-code. In practice, the scheduler is slightly more complex, since it must deal with the constraints imposed by the bandwidth allocator. Our implementation never fragments ADU's for delivery.

The `random-walk` scheduler is not tied to the specification language. The language imposes restrictions on what types of objectives can be expressed. The scheduler itself assumes nothing about the properties of individual objectives. The `random-walk` scheduler only requires a set of functions that map schedules to numeric values. To differentiate good schedules from bad ones, `evaluate_objectives` should define a partial order on the set of possible schedules.

The `random-walk` scheduler has parameters that can be adjusted to make trade-offs between scheduler accuracy and processing cost. With a `WALK_LENGTH` of 1, it becomes the randomized-round-robin scheduler. Whether accuracy is more important than cost is is likely to vary depending on the overall system.

Scheduling with Phantom Slots

A good scheduler needs look-ahead logic. In some scheduling cycles the available `txSlot`'s can be too few—or may lack critical information, that can be efficiently predicted by a channel model—to make good scheduling decisions. Imagine, for example, that at some point in time only high latency `txSlot`'s are available. If a low latency slot will be available shortly, it may be better to defer scheduling an urgent packet. Experiments demonstrate that even relatively trivial look-ahead logic boosts the accuracy of our policy driven scheduler. The alternative to predictive logic is to use infrequent scheduling cycles, increasing average queuing delays for ADU's.

Phantom transmission tokens provide a way to factor expected future channel behavior into scheduling decisions. Each channel manager can be asked to indicate, by creating phantom `txSlot`'s, how many slots it expects to have available in the near future (e.g., the next second). Predicted slots can be examined and compared in the same ways as actually available slots. With phantom slots, incorporating channel predictions into scheduling decisions does not require special-cases in the scheduler[6]. In any cycle, Horde schedulers use both normal and phantom slots to find good schedules, only transmitting as much data as can be sent using the normal `txSlot`'s. In the next cycle, the process of finding a good schedule starts afresh; the scheduler does not try to remember assignments made to phantom slots.

4 Experimental Evaluation

We have implemented a very preliminary version of the Horde middleware, and are working on building a mobile video streaming system using this implementation. In this section we report on some experimental results.

WWAN Channels

Experimental Setup Our experiments over real WWAN channels were conducted using a laptop connected to a single CDMA2000 1xRTT interface and multiple GSM/GPRS interfaces. All GPRS interfaces used the same service provider; the CDMA interface used a different provider. The GPRS interfaces were standard cell-phones connected to the laptop over a bluetooth link. The CDMA interface was a PCMCIA-based modem. The devices were in close proximity to one another and did not use specialized antennas.

The experiments reported here used three stationary WWAN interfaces. They consisted of sending as many 768 byte packets[7] as Horde's flow control layer allowed, from the laptop to a host on the MIT ethernet. We used a generic `ncManager` implementation for all channels. This implementation used AIMD for congestion control,

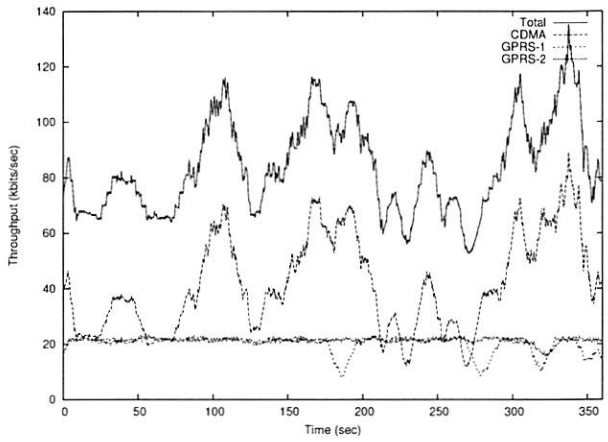


Figure 10: Throughput provided by Horde in a stationary experiment, using three colocated interfaces.

weighted-moving-averages to predict expected round-trip-times and loss rates in `txSlot`'s, and averages of past sending rates to generate phantoms.

Throughput Figure 10 shows the throughput provided in an experiment. The throughputs were calculated using a 10-second sliding window. The GPRS bandwidths are relatively stable in this experiment. This is consistent with older experiments measuring raw UDP throughput [14], which did not use Horde. However, the experiments in [14] also indicate that vehicular motion leads to significant variability in throughput. The variability in CDMA throughput may be caused by contention with other users, or due to the AIMD congestion control. Experiments in [14], when the network was—presumably—lightly loaded, have shown the CDMA interface providing upload bandwidth of up to 120 kbits/sec, close to its theoretical maximum. Generally, bandwidths on these WWAN interfaces varied greatly, both over short time scales (seen in the figure) and over longer time scales.

Packet Latency Figure 11 shows the ADU latency distributions for each channel. The Horde congestion control layer was configured to always send acknowledgments back over the lowest latency channel (the CDMA channel in this case). Consequently, the GPRS distributions do not represent GPRS packet round-trip-time distributions. Typically, those are longer and more variable.

Scheduling The existence of marked QoS differences among co-located WWAN interfaces highlights the potential impact of scheduling decisions on stream QoS. Figure 11a and 11c exemplify the very large differences

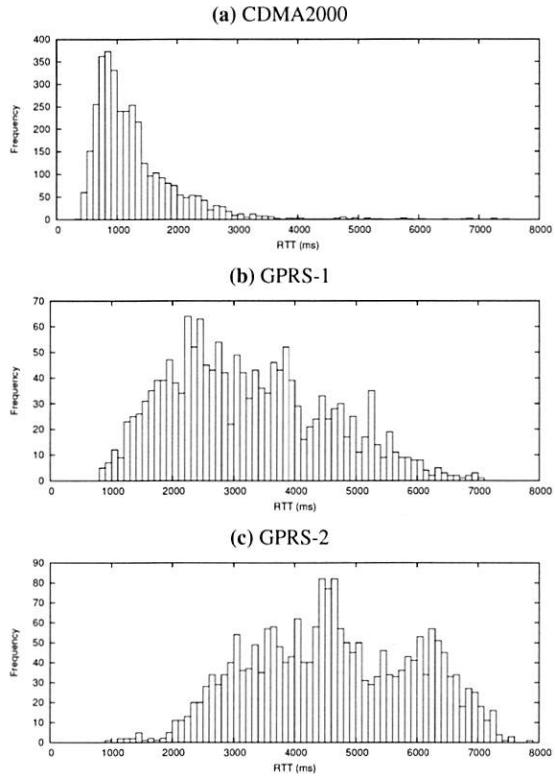


Figure 11: Observed WWAN packet RTT distributions.

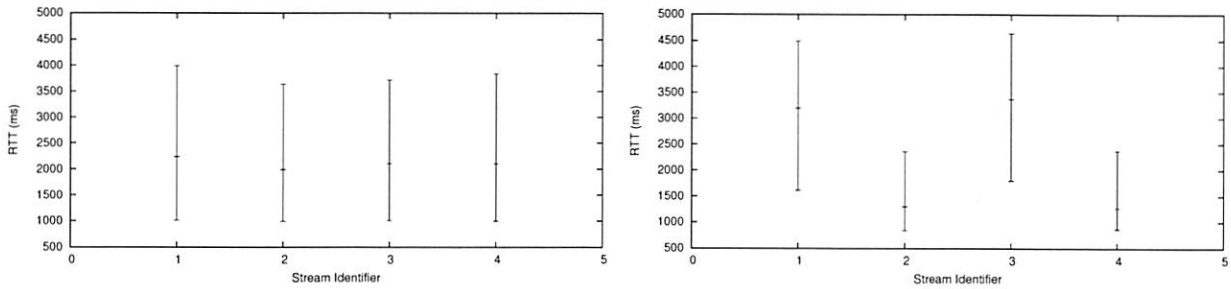
that exist between two colocated WWAN channels using different technologies and providers. Figures 11b and 11c show that even WWAN channels from the same provider using the same technology can provide very different QoS.

In order to examine the impact of different scheduling policies on QoS, we simulated the Horde scheduler on the packet traces, using simple channel models in the simulated channel managers[8]. In this way, we could compare the effects of using different schedulers and policies under identical network conditions.

Figure 12 compares a randomized round robin scheduler to Horde's `random-walk` scheduler with objectives that imply that low latency is more important for some streams. We set up four data streams, each of which demanded and received the same bandwidth.

Figure 12a shows the distribution of the latencies observed by each stream when a randomized round robin scheduler was used. As expected, there were only small variations in the latency distributions across the streams.

Figure 2b shows the distribution of observed latencies when an objective driven scheduler was used. No objective was specified for streams 1 and 3. Streams 2 and 4 were given the objective in figure 7. This objective



(a) Randomized Round Robin

(b) Objective Driven Scheduling

Figure 12: Measured round-trip-time distributions with different schedulers, striping over three existing WWANs (one CDMA and two GPRS). The graphs show the median and the upper and lower quartile packet latencies for the streams.

states that added value is derived in proportion to the difference of the expected RTT and one second. Based on this single objective, the scheduler was able to infer that it should preferentially assign transmission slots with an expected low latency to ADU's from streams 2 and 4. Figure 12b clearly shows that the scheduler recognized that QoS unfairness can be beneficial to the application.

Simulated Channels

We felt it was necessary to use simulation to fully evaluate the performance of the Horde scheduler. Our use of simulated channels was motivated by the fact that the set of real channels available to us did not provide enough QoS modulation options to the scheduler. For instance, any sort of loss or latency QoS sensitivity would drive a stream to the CDMA channel. A more diverse set of simulated channels allowed us to better investigate the performance of the scheduler.

For our simulations, we replaced the standard `ncManager` implementation with one that used pseudo-random variables to generate slots. Our simulated channels are based on measurements from real WWANs. We simulated two pairs of high-latency GPRS channels (each pair had a different mean latency); one medium-latency low-loss CDMA channel; and one low-latency medium-loss channel based on CDMA statistics.

We injected objectives for streams 1 and 4. Two objectives were defined for stream 4: one valuing low-latency and one valuing low-loss. A single low-latency objective was defined for stream 1. Figure 13 shows the resulting latency distributions for the streams. Streams 2 and 3 are spread over all channels, indicated by their large latency ranges; stream 4 mostly uses the medium-latency low-loss channel; and stream 1 is mostly spread over the medium-latency and low-latency channels.

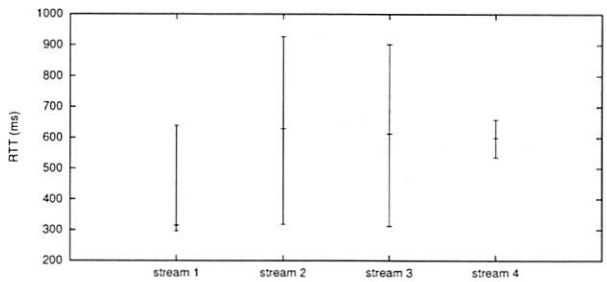
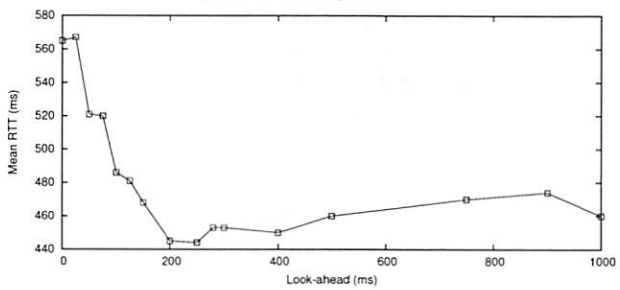


Figure 13: RTT distributions in a simulated experiment.

(a) $50ms$ cycle period



(b) $500ms$ cycle period

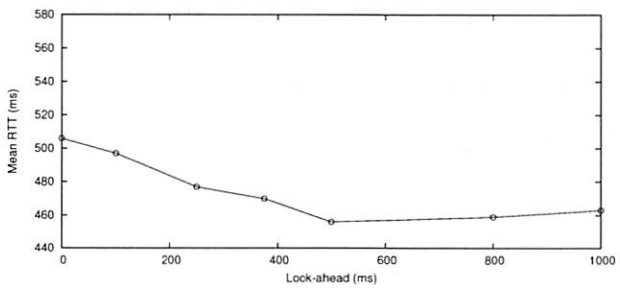


Figure 14: Impact of look-ahead on accuracy. The graphs show how the average latency on a latency sensitive stream varied with different look-ahead and scheduling cycle periods, on simulated channels.

Effects of Look-ahead on Accuracy

We were interested in how important look-ahead was for an accurate objective driven scheduler. To investigate this, we injected a single objective valuing a low network-latency for ADU's on a specific stream. We ran the scheduler with a number of different cycle periods (T) and different look-ahead periods (L), on simulated channels. In these experiments, a lower average network latency for the latency-sensitive stream implied a more accurate scheduler. Figure 14 shows how latency varied for some different values of T and L.

Scheduler accuracy improves by increasing L (up to some L). With a larger look-ahead period the scheduler can see, in each cycle, the future slots from both the low-latency high-rate channels (CDMA) and the high-latency low-rate channels (GPRS). Thus with a large enough L (around $200ms$) the scheduler works well.

Scheduler accuracy also improves with T (up to some value of T). Again the scheduler is able to see both types of slots in each cycle, improving its accuracy. However, with a higher T, ADU's are queued longer locally, so the overall ADU delay becomes quite large, even as the scheduler becomes more accurate.

5 Related Work

Schemes that provide link layer striping have been around for a while [9, 8, 1]. Most such schemes were pioneered on hosts connected to static sets of ISDN links. These schemes mostly assume stable underlying channels. Instabilities and packet losses can cause some protocols to behave badly [1]. Usually IP packets are fragmented and distributed over the active channels, if one fragment is lost, the entire IP packet becomes useless. This results in a magnified loss rate for the virtual channel. Horde avoids this magnification by exposing ADU fragment losses to the application, consistent with the ALF principle.

Most link-layer striping systems are, quite reasonably, optimized for TCP. Mechanisms are chosen to minimize reordering, and packets may be buffered at the receiver to prevent the TCP layer from generating DUPACKS [1]. partial IP losses can be exposed to the higher layer by rewriting TCP headers within the striping code.

Implementing congestion control under the striping scheduler is not a new idea. The LQB scheme [18] extends the deficit-round-robin approach to reduce the load on channels with higher losses. The use of TCP-v [10], a transport layer protocol that provides network striping with the goal of achieving reliable in-order delivery, also results in per-channel congestion control. However, whereas Horde allows channel-specificity, others use the same mechanism on every type of channel.

Some recent proposals for network striping over wireless links have proposed mechanisms to adjust striping policy. Both MAR [15] and MMTP [11] allow the core packet scheduler to be replaced with a different scheduler implementing a different policy. This is the hard-coded policy case mentioned earlier.

We were motivated by the need to develop a multi-path video streaming application. Setton et al [16] provide an example of such an application that uses multiple-descriptions of the source video, spreading them out over the available paths, based on a binary-metric quality of the paths. Begen et al [5] describe a similar scheme. Horde allows more flexible scheduling of video packets over the available paths.

6 Summary and Conclusion

Horde is a library that facilitates flexible network striping in WWAN environments for a diverse range of applications. Horde separates the striping policy from routing and scheduling. It allows applications to describe QoS objectives for each stream that the striping mechanism attempts to satisfy. Horde can be used by a set of application data streams, each with its own policy, to flexibly stripe data over a highly heterogeneous set of dynamically varying network channels.

Horde is most useful when different streams gain value from different aspects of network performance and when the available network channels have dissimilar and/or time-varying characteristics.

Horde tackles many complex problems. We have presented techniques to separate striping policy from the mechanism used, and evaluated how well a generic striping mechanism is able to interpret simple expressed policies. Furthermore, Horde has been designed from the ground-up with the assumption that the set of channels being striped over are not well-behaved and can have significantly different channel characteristics

In this paper, we described the basic abstractions upon which Horde is built. The two key abstractions are transmission slots (`txSlot`'s) and policy *objective*'s. The `txSlot`'s are generated by Horde to abstractly capture the expected short-term performance of the network channels. The objectives are written by application programmers to abstractly specify network-QoS related objectives for individual data streams. The Horde scheduler uses `txSlot`'s and objectives to derive transmission schedules that provide better value to applications than do conventional scheduling algorithms.

Experiments with our initial Horde implementation confirm our belief that the kind of QoS modulation Horde aims to achieve is both realistic and advantageous to actual applications.

In conclusion, more work is needed on the specification language and schedulers. Presently, the language allows too much complexity, making objective programming harder than it should be. Furthermore, some abstractions could be generalized more (e.g., streams, and correlated loss) as operations over sets of ADUs. We are working on addressing these issues. Finally, we are experimenting with more complex schedulers that use language semantics to derive better schedules.

Acknowledgments

The authors would like to thank Allen Miu, Godfrey Tan, Magdalena Balazinska, Vladimir Bychkovsky, Eugene Shih, Michel Goraczko and Dorothy Curtis. This research is supported by the National Library of Medicine, CIMIT, the Center for the Integration of Medicine and Innovative Technology, and Acer Inc., Delta Electronics Inc., HP Corp., NTT Inc., Nokia Research Center, and Philips Research under MIT Project Oxygen.

Notes

[1]The IPC overhead is acceptable in our system, since the aggregate data rates are low (300 kbits/sec). The IPC stubs can be easily replaced with more efficient mechanisms if needed, for a single-process Horde.

[2]With multiple destination hosts, a `ncManager` may need to maintain a unique congestion control session for every destination.

[3]Moving-averages seemed to work reasonably well in our experiments, though they were often wrong by over $100ms$. We found better estimators could be constructed using *a priori* knowledge about the WWAN technology [14]. Furthermore, more elaborate estimators plug easily into our modular framework: a `ncManager` that uses signal strength readings to estimate a loss probability, or one that uses an accelerometer input to predict when a motion sensitive channel will have more losses, could be incorporated without complications.

[4]If fairness is of concern, this definition can be modified to take that into account. In this paper, we restrict our consideration of fairness to bandwidth allocations.

[5]Nonetheless, since a stream may receive slots more likely to result in losses, that stream may not receive its fair share of goodput. Fair shares of slots do not imply that all streams are allocated equal goodput.

[6]Depending on the scheduler design, the constraints imposed by the `bwAllocator` can introduce wrinkles in how the scheduler uses phantom slots. In summary: the scheduler must ensure that, in the long run, the allocations are met even though phantoms may cause short-term deviations.

[7]Our UDP experiments on WWAN channels [14] show that packet size affects both available throughput and packet latency. Generally: smaller packets experience lower latency; larger packets yield higher raw UDP throughput. 768 byte packets seem to be a good compromise

[8]In the `txSlot` expectations, expected packet latency was calculated using a weighted moving average of known packet latencies. Loss probabilities were, similarly, averages.

References

[1] ADISESHU, H., PARULKAR, G. M., AND VARGHESE, G. "A Reliable and Scalable Striping Protocol". In *SIGCOMM* (1996), pp. 131–141.

[2] ANDERSEN, D., BANSAL, D., CURTIS, D., SESHAN, S., AND BALAKRISHNAN, H. "System support for bandwidth management and content adaptation in Internet applications". In *Proceedings of 4th Symposium on Operating Systems Design and Implementation, USENIX* (October 2000), pp. 213–226.

[3] APOSTOLOPOULOS, J., AND WEE, S. "Unbalanced Multiple Description Video Communication using Path Diversity". citeseer.ist.psu.edu/apostolopoulos01unbalanced.html.

[4] BALAKRISHNAN, H., PADMANABHAN, V. N., SESHAN, S., AND KATZ, R. H. "A comparison of mechanisms for improving TCP performance over wireless links". *IEEE/ACM Transactions on Networking 5*, 6 (1997), 756–769.

[5] BEGEN, A., ALTUNBASAK, Y., AND ERGUN, O. "Multi-path Selection for Multiple Description Encoded Video Streaming". In *IEEE ICC* (2003).

[6] CARTWRIGHT, J. "GPRS Link Characterisation". http://www.cl.cam.ac.uk/users/rc277/linkchar.html.

[7] CLARK, D., AND TENNENHOUSE, D. "Architectural Consideration for a New Generation of Protocols". In *ACM SIGCOMM* (1990).

[8] DUNCANSON, J. "Inverse Multiplexing". *IEEE Communications Magazine* (April 1994), 34–41.

[9] FREDETTE, P. "The Past, Present, and Future of Inverse Multiplexing". *IEEE Communications Magazine* (April 1994), 42–46.

[10] HSIEH, H., KIM, K., AND SIVAKUMAR, R. "a Transport Layer Approach for Achieving Aggregate Bandwidths on Multi-homed Hosts". In *ACM MOBICOM* (2002), pp. 83–94.

[11] MAGALHAES, L., AND KRAVETS, R. MMTP: Multimedia multiplexing transport protocol, 2001.

[12] MAGALHAES, L., AND KRAVETS, R. "Transport Level Mechanisms for Bandwidth Aggregation on Mobile Hosts". In *ICNP* (2001).

[13] PERLOFF, J. M. "Microeconomics 3rd edition" . Addison Wesley Publishing Company.

[14] QURESHI, A. Flexible Application Driven Network Striping over Wireless Wide Area Networks. MEng Thesis, Massachusetts Institute of Technology, March 2005.

[15] RODRIGUEZ, P., CHAKRAVORTY, R., CHESTERFIELD, J., PRATT, I., AND BANERJEE, S. MAR: A Commuter Router Infrastructure for the Mobile Internet. In *Mobisys* (2004).

[16] SETTON, E., LIANG, Y. J., AND GIROD, B. Multiple description video streaming over multiple channels with active probing. In *IEEE International Conference on Multimedia and Expo* (2003).

[17] SINHA, P., VENKITARAMAN, N., SIVAKUMAR, R., AND BHARGHAVAN, V. "WTCP: A Reliable Transport Protocol for Wireless Widea-Area Networks". University of Illinois at Urbana-Champaign.

[18] SNOEREN, A. "Adaptive Inverse Multiplexing for Wide-Area Wireless Networks". In *IEEE conference on Global Communications* (1999), pp. 1665–1672.

An Overlay MAC Layer for 802.11 Networks

Ananth Rao
UC Berkeley

Ion Stoica
UC Berkeley

Abstract

The widespread availability of 802.11-based hardware has made it the premier choice of both researchers and practitioners for developing new wireless networks and applications. However, the ever increasing set of demands posed by these applications is stretching the 802.11 MAC protocol beyond its intended capabilities. For example, 802.11 provides no control over allocation of resources, and the default allocation policy is ill-suited for heterogeneous environments and multi-hop networks. Fairness problems are further exacerbated in multi-hop networks due to link asymmetry and hidden terminals. In this paper, we take a first step towards addressing these problems *without* replacing the MAC layer by presenting the design and the implementation of an *Overlay MAC Layer* (OML), that works on top of the 802.11 MAC layer. OML uses loosely-synchronized clocks to divide the time in to equal size slots, and employs a distributed algorithm to allocate these slots among competing nodes. We have implemented OML in both a simulator and on a wireless testbed using the Click modular router. Our evaluation shows that OML can not only provide better flexibility but also improve the fairness, throughput and predictability of 802.11 networks.

1 Introduction

In recent years, the popularity of the 802.11 protocol has made it the de facto choice for developing and deploying a multitude of wireless networks and applications. In addition to the traditional model of a last-hop wireless link to an access point, 802.11 networks are used to setup wireless infrastructures in corporate networks, long-haul links using directional antennae in rural areas [31], and more recently multi-hop networks for broadband Internet access, *e.g.,* rooftop or mesh networking [5, 20].

Despite making deployment easier, the 802.11 protocol does pose serious limitations in addressing the different demands of these emerging applications. The 802.11 MAC protocol was carefully engineered for the wireless LAN environment [27] and many of the underlying assumptions may not hold in the new environments. Several problems have been reported in earlier research [23]. We illustrate more problems through a careful performance study of the MAC using statistics collected from the device driver and packet-sniffing tools [1, 2]. In particular, we show that link asymmetry or hidden terminals can either cause (a) poor fairness, sometimes even shutting off all flows through a node or (b) excessive collisions, leading to poor performance. In addition, we consider several scenarios where the default allocation of the medium by 802.11 is far from being optimal.

Two main approaches have been proposed in the literature to address these problems. The first approach is to build workarounds in the routing or transport layers to avoid the cases where the MAC layer performs badly [5, 20, 22, 38]. The second approach focuses on replacing the MAC layer with new protocols [10, 16] and standards such as 802.16 (WiMax) and 802.11e. The first approach is more easily deployable as the functionality of both the routing and transport layers is fully implemented in software, so it is relatively easy to modify. However, as we will show in this paper, the 802.11 MAC layer suffers from certain limitations, such as unfairness due to asymmetric interference, that cannot be fully addressed through changes only to the higher layers. In contrast, the second approach is far more powerful since modifying the MAC layer can directly address all the 802.11 limitations, but it is much harder to implement and experiment with. The MAC layer is implemented partly in hardware, partly in firmware, and partly in the device driver of the Network Interface Card (NIC). Thus, changing the MAC layer can require hardware and firmware changes, a highly expensive proposition.

In this paper, we propose a third approach that combines power of changing the MAC layer and the ease of deployability of modifying only the higher layers. In particular, we propose the design of an Overlay MAC layer (OML) on *top* of the 802.11 MAC layer. OML performs access control and scheduling, and does not require any changes to the 802.11 MAC hardware or the 802.11 standard. Our approach is inspired by the success of the "Overlay" networks, which have been used in the past few years to study new network protocols and implement new functionality without any modifications to the underlying IP layer. In the context of the MAC layer, using an overlay offers the following three advantages. First, it provides an immediately useful piece of software which can be used in 802.11 networks while waiting for newer standards to become more widespread. Second, the additional flexibility of having the MAC layer in software allows better integration with routing and application requirements. Third, it allows research on new protocols to be conducted on any of the numerous, already-deployed 802.11 testbeds. However, these advantages do not come for free. Like many overlay approaches, OML suffers

some additional overhead compared to implementing the same changes at the MAC layer. In addition, the design of OML is limited by the interface exposed by the 802.11 MAC layer. For example, OML cannot carrier sense the communication channel since 802.11 network cards do not typically export the channel status to higher layers. Despite such limitations, we believe that the advantages of the our overlay approach make it a valuable alternative to modifying the MAC layer.

To address the previously mentioned limitations of the 802.11 MAC layer, OML uses loosely synchronized clocks to divide the time in equal size slots, and then uses a distributed algorithm to allocate these slots across the competing nodes. In addition to preventing unfavorable interaction between senders at the underlying MAC layer, OML also allows users to implement application-specific resource allocation in the same way overlay networks allow application-specific routing. The slot allocation algorithm of OML, called Weighted Slot Allocation (WSA), implements the weighted fair queueing policy [19], where each of the competing nodes that has traffic to send receives a number of slots proportional to its weight.

The main contribution of this paper is the architecture and implementation of the Overlay MAC Layer. We believe this is the first time such an approach has been proposed to improve the performance and flexibility of the MAC layer. Because of the ease of deployability of the overlay approach, we are able to demonstrate these improvements through a real implementation of OML on a testbed. OML not only offers better flexibility than the 802.11 MAC layer, it also improves throughput and predictability by minimizing losses due to contention.

The rest of the paper is organized as follows. Section 2 presents the related work, and Section 3 uses experimental results to motivate the need for an overlay MAC protocol. Section 4 describes the challenges in designing OML. Section 5 describes the hardware and software of our testbed. Sections 6 and 7 present simulation and experimental results. Finally, Section 8 discusses the open issues and limitations, and Section 9 concludes the paper.

2 Related Work

802.11 MAC Limitations: Many researchers have reported problems with the 802.11 MAC protocol. Heusse *et al.* have described how senders with heterogeneous data rates can affect the system throughput adversely [23]. The Roofnet and Grid projects [4, 5] have reported a variety of problems with 802.11 multi-hop testbeds such as low throughput and unpredictable performance. We add to the set of the problems reported in these studies, by showing that *asymmetric* interference at the MAC layer can cause significant unfairness.

MAC Layer Improvements: A plethora of MAC protocols have been proposed to improve the predictability and the resource management capabilities of wireless networks. Many of these solutions use either sophisticated back-off protocols [12] or slot allocation algorithms [10, 11] to implement more flexible allocation policies such as Weighted Fair Queueing [19]. Other solutions achieve this goal by using a combination of both back-off and slot allocation algorithms [16, 30]. All these protocols work at the MAC layer and assume full control over the hardware and the physical layer. In contrast, we assume that OML can use only the limited interface exposed by most 802.11 cards to control packet transmission.

Routing and Transport Layer Improvements: Some recent proposals use mechanisms above the MAC layer to improve the fairness of 802.11 networks. Gamrioza *et al.* [22] propose to explicitly limit the sending rate of the flows nodes, while Yi and Shakkottai [38] propose to implicitly limit the TCP flow rates by delaying the TCK ACKs at intermediate nodes to achieve fair bandwidth allocation. However, as we show in Section 3.2, controlling only the sending rate is not enough to avoid all undesirable interaction between competing flows.

Our work is complementary to several proposals that aim to improve the performance of 802.11 multi-hop networks. For example, Couto *et al.* [18] propose a new routing metric to improve network throughput. OML can be used in conjunction or integrated with such routing protocols to improve the throughput or other network metrics. Extremely Opportunistic Routing [13] (ExOR) proposes a modification to the 802.11 MAC so that any eligible node that receives a packet without errors can send an ACK and forward the packet. Since OML schedules transmissions based on the sender only, and not on the destination of a packet, we believe that OML can be used on top of ExOR to obtain better performance.

Distributed Slot Allocation: The Weighted Slot Allocation (WSA) algorithm that we propose in this paper is based on the Neighborhood-aware Contention Resolution (NCR) protocol, which was previously proposed by Bao and Garcia-Luna-Aceves [10, 11]. WSA extends NCR in two aspects. First, unlike OML, NCR assumes that the interference graph in a multi-hop network consists of isolated, easily identifiable cliques, *i.e.,* if node A interferes with B and B with C, then A interferes with C. Second, while NCR uses *pseudo-identities* to support integer weights, WSA can support arbitrary weights.

Overlay Networks: Our solution is similar in spirit to the overlay network solutions that aim to improve routing resilience and performance in IP networks [8, 9, 15, 36]. Overlay networks try to overcome the barrier of modifying the IP layer by employing a layer on top of the IP to implement the desired routing functionality. Simi-

larly, OML runs on top of the existing 802.11 MAC layer, and its goal is to enhance the MAC functionality without changing the existing MAC protocols.

3 Motivation

In this section, we motivate our approach of building an Overlay MAC Layer for 802.11 networks. Briefly, our motivation stems from the following three factors:

1. Due to *asymmetric interaction* between flows and an *inflexible default allocation policy*, the efficiency and fairness of the 802.11 MAC suffer in a number of scenarios.
2. Solutions that only modify layers above the MAC, though applicable in certain cases, are of limited use in addressing certain undesirable interactions at the MAC layer.
3. The additional flexibility, the low cost and the immediate deployability of an overlay solution makes it an attractive alternative for developing a new MAC layer or modifying an existing one.

Next, we substantiate the first two factors by conducting experiments on a six node wireless testbed using 802.11a radios. In Section 3.1, we illustrate the limitations of 802.11 in the form of asymmetric interference and the inflexible default allocation policy, and in Section 3.2 we argue that these limitations cannot be fully addressed at layers above MAC. For more details on our testbed, please refer to Section 5.

3.1 Limitations of 802.11 MAC Layer

In this section, we illustrate two specific limitations of the 802.11 MAC protocol using simple experiments:

1. *Asymmetric interactions:* Interference between two flows either at the senders or at the receivers can cause one flow to be effectively shut-off.
2. *Sub-optimal default allocation:* As other researchers have pointed out [23], we show that the default allocation of the transmission medium by the MAC layer fails to meet the requirements of some applications.

3.1.1 Effect of Asymmetric Interaction

In this section, we take a closer look at the impact of interference on performance at the MAC layer. Interference has been often cited as being the cause for poor performance in other testbeds [5]. Here, we try to understand and quantify the effect of interference through experiments on a multi-hop testbed. To avoid any complex interaction between the MAC, routing, and transport layers, we restrict our experiments to two simultaneous 1-hop UDP flows. Figure 1 shows the location of machines in our testbed in relation to the floor plan of our office building. We also show the signal strength of each link in either direction as reported by the device driver of the wireless card. The topology of the testbed resembles a chain and we study how simultaneous transmissions along two links in the chain interfere with each other.

1. *Asymmetric sender interaction:* Because of asymmetry in radio propagation, one sender may be able to carrier sense transmissions from the other sender, but not vice-versa. This can lead to starvation of the first sender.
2. *Asymmetric receiver interaction:* Hidden terminal problems can completely shut-off flows in the network, particularly in the presence of other flows which do not experience similar problems.

Asymmetric Carrier Sense: We conduct our first set of experiments using broadcast packets only. This allows us to better understand the interaction between *senders*, as broadcast communication abstracts away the ACKs and packet retransmissions at the MAC layer. We make the two senders continuously send broadcast UDP packets and measure the sending rate of each sender averaged over 1 minute. The sending rate of each node depends only on whether it can carrier sense transmissions from the other node.

We conducted this experiment for each of the 15 pairs of nodes in our six node testbed. As expected, nodes that were far away (*e.g.,* nodes 1 and 5 in Figure 1) did not carrier sense each other and were able to simultaneously send at about 5.1 Mbps. When the nodes are close to each other (*e.g.,* nodes 2 and 3), they carrier sense each others transmissions, and hence share the channel capacity to send at about 2.5 Mbps each. However, in *three* cases we found than one of the nodes was transmitting at more than 4.5 Mbps, while the other was transmitting at less than 800 Kbps. The only possible explanation for this asymmetric performance is that one sender can carrier sense the other, but not vice-versa. This can impact all configurations of flows where both senders are active, irrespective of who the receivers are.

Asymmetric Receiver Interaction: Now, we study the interference at the receiver. To eliminate the effects of sender (carrier sense) interference, we only consider senders that are able to broadcast simultaneously at full rate. In these cases, we conducted experiments with each sender sending unicast UDP packets simultaneously to receivers within their communication range at the maximum rate. We found that depending on the configuration of the flows, either one or both receivers can be affected by interference. For example, when the flows from node 1 to 2 and from node 3 to 4 are simultaneously active, node 2 experiences interference from node 3, whereas the latter flow is not disrupted by transmissions from node 1. We

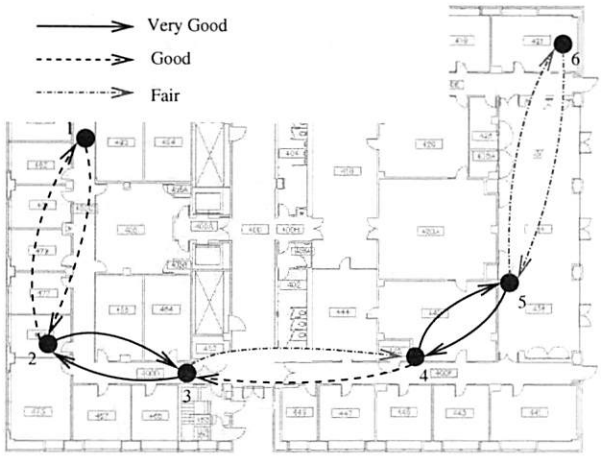


Figure 1: Location of the nodes and the signal strength of each link as reported by the driver in our multi-hop testbed

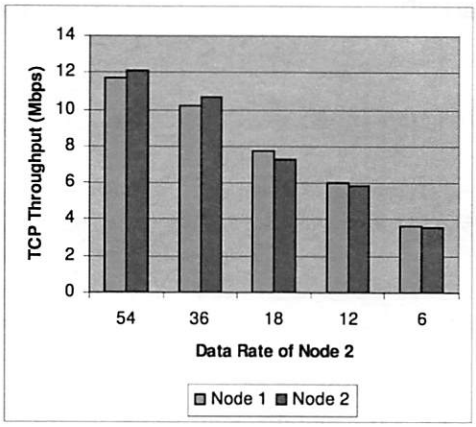


Figure 2: 802.11 throughput in the presence of heterogeneous data rate senders

found two such cases where the sending rate of affected sender drops by more than 60% due to the repeated back-off at the MAC layer triggered by retransmission timeouts. Furthermore, based on the statistics gathered from the device driver, we found more than 85% of the packets sent by this sender were not received at the destination. In the second case, both flows can experience problems due to interference, *e.g.,* when both nodes 1 and 3 are sending to node 2. This case illustrates the classic hidden terminal problem, where back-off at the MAC layer causes the sending rate of both senders to drop, which in turn reduces the loss rate of both flows to 35%. However, in this case, the channel utilization and hence the total system throughput drops by about 55%.

3.1.2 Sub-optimal Default Allocation

In this section, we show two examples where the default bandwidth allocation by the 802.11 MAC is far from ideal. These examples illustrate the need for a flexible allocation policy which can be controlled by applications.

Heterogeneous Transmission Rates: The 802.11 MAC allocates an equal number of *transmission opportunities* to every competing node. However, as shown in previous work [23], this fairness criterion can lead to a low throughput when nodes transmit at widely different rates.

We illustrate this behavior using a simple experiment comprising two heterogeneous senders connected to a single access point. We emulate heterogeneous senders by fixing the data-rate of transmissions from Node 1 to the access point at 54 Mbps and varying the data-rate of Node 2 between 6 Mbps and 54 Mbps. Figure 2 shows the average throughput of two TCP flows originated at the two nodes. This experiment shows that while the behavior is fair as nodes see equal performance irrespective of their sending rate, it hurts the overall system throughput. In particular, as the sending rate of Node 2 decreases from 54 Mbps to 6 Mbps, the total system throughput decreases from 24 Mbps to 7.2 Mbps. In addition, this behavior leads to poor predictability. For example, if Node 1 is the only active one, it will have a throughput of roughly 24 Mbps. However, when Node 2 starts transmitting at 6 Mbps, the throughput seen by Node 1 drops to 3.6 Mbps.

Several Flows Traversing a Node: In a multihop network, the fairness policy implemented by 802.11 can lead to poor fairness. This is because in an ad-hoc network, the fairness policy of 802.11 does not account for the traffic forwarded by a node on the behalf of other nodes. Ideally, we would like to provide higher priority to nodes that relay traffic from a lot of flows.

3.2 Need for a MAC Layer Solution

A natural question that arises is whether the limitations we described so far can be addressed at a layer above the MAC layer, such as the network or transport layer. Several proposals have tried to answer this question affirmatively [22, 38]. In a nutshell, these proposals estimate the capacity and interference patterns of the network, and then use this information to limit the sending rate (usually, employing token-buckets) of each node at the network layer. While these solutions may perform well in many scenarios, our experience suggests that in certain practical situations they fall short. For instance, consider the problem of fair allocation as defined in [22]. In order to achieve fair allocation, the following two requirements should hold:

1. Every node should access the medium for only its

fair share of time.

2. When a node is transmitting data, other nodes should not interfere with it.

By limiting the sending rate of each node according to its fair share, we can address the first requirement. However, this solution would work only if the underlying MAC layer satisfies the second requirement. Unfortunately, as discussed in Section 3.1, the 802.11 MAC fails to satisfy this requirement quite often. As an example, in the chain configuration of our testbed, we found that both the first and the third link of the chain taken in isolation had a loss rate of less than 5% and were able to support a TCP flow at close to the channel capacity of 4.6 Mbps. But when we started simultaneous flows on both links, one flow always received less than 100 Kbps whereas the other flow received in the excess of 4 Mbps. In order to mitigate this problem we tried rate limiting both TCP flows to 2.3 Mbps. In this case we found that the throughput of the first flow only improved to about 580 Kbps even though the other flow was only receiving 2.3 Mbps (as specified by the token bucket).

The inability of 802.11 to effectively address the second requirement suggests that a general solution to fully address all the 802.11 limitations requires changes to the MAC layer.

3.3 Advantages of an Overlay Solution

Given that the 802.11 MAC limitations cannot be fully addressed without changing the MAC layer, we propose the use of an Overlay MAC layer as an alternative to building a new MAC layer. We believe that the overlay approach offers several advantages such as low cost, flexibility and the possibility of integration with higher layers.

First, changing the MAC layer requires the use of expensive proprietary hardware, or waiting for a new standard and hardware to become available. In contrast, OML can be deployed using the existing 802.11 hardware.

Second, the fact that OML is implemented in software makes it easier to modify OML to meet the diverse requirements of the ever increasing spectrum of wireless applications [4, 5, 23, 31]. For example, in our OML implementation, we support service differentiation both at the flow and node granularity. In the case of a rooftop network [17], one could modify OML to take advantage of the relative stationarity of the link quality and interference patterns.

Finally, the software implementation of OML also enables us to have tighter integration between the link, network and transport layers. For example, Jain *et al* [24] show that it is possible to significantly increase the throughput of an ad-hoc network by integrating the MAC and routing layers. Another example is that the transport layer can provide information about the traffic type (*e.g.*, voice, ftp, web), and OML can use this information to compute efficient transmission schedules.

Of course, these advantages do not come for free. Fundamentally, OML incurs a higher overhead and it is more inefficient than a hardware implementation of the same functionality. Furthermore, OML is limited to using only the interface exposed by the 802.11 MAC layer as opposed to all the primitives that the hardware supports. We present some of the limitations of OML in more detail in Section 8.

4 Design

In this Section, we present our solution, an overlay MAC layer (OML), that alleviates the MAC layer issues described in Section 3. We first state our assumptions.

4.1 Assumptions

The primitives available for the design of OML are determined by the interface exposed by the device driver. For the sake of generality, in this paper we make minimal assumptions about this interface. In particular, we assume that

1. The card can send and receive both unicast and broadcast packets.
2. It is possible to set the wireless interface in promiscuous mode to listen to all transmissions from its 1-hop neighbors.
3. It is possible to disable the RTS-CTS handshake by correspondingly setting the RTS threshold.
4. It is possible to limit the number of packets in the card's queue to a couple of packets. This is critical for enabling OML to control packet scheduling because once the packets are in the card's queue, OML has no control over when these packets are sent out.

We note that these assumptions already hold or can be enforced, eventually by modifying the device drivers in most 802.11 cards.

While carrier-sensing is a very important primitive for the design of a MAC protocol, we do not assume that OML can use this primitive. This assumption reflects the fact that most 802.11 cards do not export the current status of the channel or the network allocation vector (NAV)[1] to the higher layers.

4.2 Solution

As noted in the previous section, the only control that OML can exercise over packet scheduling is *when* to send a packet to the network card. Once the packet is enqueued at the network card, OML has no control on when the packet is actually transmitted. Thus, ideally, we would

like that when OML sends a packet to the network card, the network card transmits the packet immediately (or at least with a predictable delay).

To implement this idealized scenario, we propose a solution that aims to (a) limit the number of packets queued in the network card, and (b) eliminate interference from other nodes, which is the major cause of packet loss and unpredictability in wireless networks. Goal (a) can be simply achieved by reducing the buffer size of the network card.

To achieve goal (b), we synchronize clocks and we use a TDMA-like solution where we divide the time into slots of equal size l, and allocate the slots to nodes according to a weighted fair queueing (WFQ) policy [19]. We call this allocation algorithm the Weighted Slot Allocation (WSA) algorithm. WSA assigns a weight to each node, and in every interference region[2] allocates slots in proportion to nodes' weights. Thus, a node with weight two will get twice as many slots as a node with weight one in the same interference region. Only nodes that have packets to send contend for time slots, and a node can transmit only during its time slots. Since a time slot is allocated to no more than one node in an interference region, no two sending nodes will interfere with each other. This can substantially increase the predictability of packet transmission, and reduce packet loss at the MAC layer. However, in practice, it is hard to totally eliminate interference. In an open environment there might be other devices out of our control (*e.g.,* phones, microwave ovens), as well as other nodes that run the baseline 802.11 protocol which can interfere with our network. Furthermore, as we will see, accurately estimating the interference region is a challenging problem.

The reason we base WSA on the WFQ policy is because WFQ is highly flexible, and it avoids starvation. WFQ has emerged as the policy of choice for providing QoS and resource management in both network and processor systems [19, 28, 37]. However, note that WSA is not the only algorithm that can be implemented in OML; one could easily implement other allocation mechanisms if needed.

There are three questions we need to answer when implementing WSA: (a) what is the length of a time-slot, l, (b) how are the starting times of the slots synchronized, and (c) how are the times slots allocated among competing clients. We answer these questions in the next three sections.

4.2.1 Slot size

The slot size l is dictated by the following considerations:

1. l has to be considerably larger than the clock synchronization error.
2. l should be larger than the packet transmission time, *i.e.,* the interval between the time the first bit of the packet is sent to the hardware, and the time the last bit of the packets is transmitted in the air.
3. Subject to the above two constraints, l should be as small as possible. This will decrease the time a packet has to wait in the OML layer before being transmitted and will decrease the burstiness of traffic sent by a node.

In our evaluation, we chose l to be the time it takes to transmit about 10 packets of maximum size (*i.e.,* 1500 bytes). We found this value of l to work well in both simulations and in our implementation.

4.2.2 Clock synchronization

Several very accurate and sophisticated algorithms have been proposed for clock synchronization in multi-hop networks [21, 33, 35]. We believe that it is possible to adapt any of these algorithms to OML. However, OML does not require very precise clock synchronization since we use a relatively large slot time. For evaluation of OML, we have implemented a straightforward algorithm that provides adequate performance within the context of our experiments on the simulator and the testbed.

We synchronize all the clocks in the network to the clock at a pre-designated *leader node*. We estimate the one-way latency of packet transmission based on the data-rate, packet size and other parameters of the 802.11 protocol. We then use this estimated latency and a timestamp in the header of the packet to compute the clock skew at the receiver.

4.2.3 Weighted Slot Allocation (WSA)

In this section, we describe the design of WSA. The challenge is to design a slot allocation mechanism that (a) is fully decentralized, (b) has low control overhead, (c) and is robust in the presence of control message losses and node failures.

For ease of explanation, we present our solution in three stages. In the first two stages, we assume that any two nodes in the network interfere. Thus, only one sender can be active in the network at any given time. Furthermore, in the first stage, we assume that all nodes have unit weight. In the final stage, we relax both these assumptions.

Step 1 - Network of diameter one with unit weights: One solution to achieve fair allocation is to use pseudo-random hash functions as proposed in [11]. Each node computes a random number at the beginning of each time slot, and the node with the highest number wins the slot. The use of pseudo-random hash functions allow a node to compute not only its random number, but the random

numbers of others nodes without any explicit communication with those nodes.

Let H be a pseudo-random function that takes values in the interval $(0, 1]$. Consider c nodes, n_1, n_2, ..., n_c, that compete for time slot t. Then each node computes the value $H_i = H(n_i, t)$ for $1 \leq i \leq c$, where H is a pseudo-random function, and t is the index of the slot in contention. A node n_r wins slot t if it has the highest hash value, *i.e.,*

$$\arg\max_{1 \leq i \leq c} H_i = r. \quad (1)$$

Since H_i is a pseudo-random random number, it is equally likely that any node will win the slot. This results in a fair allocation of the time slots among the competing nodes.

A node n_s will incorrectly decide that it can transmit if and only if it is unaware of another node n_i such that $H_i > H_s$. While the probability that this can happen cannot be neglected (*e.g.,* when a node n_i joins the network), having more than one winner occasionally is acceptable as the underlying MAC layer will resolve the contention using CSMA/CD. As long as such events are rare, they will not significantly impact the long term allocation.

Step 2 - Network of diameter one with arbitrary weights: Let w_i denote an arbitrary weight associated to node i. Then we define $H_i = H(n_i, t)^{1/w_i}$, and again allocate slot t to node r with the highest number H_r. The next result shows that this allocation will indeed lead to a weighted fair allocation.

Theorem 1. *If nodes n_1,...,n_c have weights w_1,...,w_c, and H is a pseudo-random function that takes values in the range $(0, 1]$, then*

$$P[(\arg\max_{1 \leq i \leq c} H_i) = r] = \frac{w_r}{\sum_{j=1}^{c} w_j} \quad (2)$$

Proof. Due to space limitations, we refer the reader to our technical report [32] for the proof.

Step 3 - Larger diameter network: To enable frequency reuse in networks of a larger diameter, WSA must be able to assign the same slot to multiple nodes as long as they do not interfere with each other. Ideally, the decision to allocate a new time slot should involve only nodes that interfere with each other. Therefore, a given node n should use only the hash values computed for nodes in its interference region. Note that WSA will only ensure weighted fairness among the nodes in the same interference region. Nodes in different interference regions can get very different allocations, based on the level of contention in their regions.

Computing the set of nodes that interfere with a given node is a hard problem, and we are not aware of any good distributed algorithm to solve it. We get around this problem by simply assuming that a node can interfere with *all* nodes within k-hop distance, where k is given. When a node wants to contend for a slot it broadcasts its intention to all nodes within k hops. The set of nodes form which a node hears a broadcast message is then the set of nodes it assumes it interferes with.

There is a clear trade-off between the probability of interference and the bandwidth utilization in choosing the value of k. As the value of k increases, both the probability of interference and the utilization decrease. The reason why the utilization decreases is because, as k increases, a k-hop region will cover more and more nodes that do not interfere with each other in the real system. In this paper, we assume two values of k, k=1, which represents an optimistic assumption as the interference range is typically greater than the transmission range, and k=2, which as we found in our experiments is a more conservative assumption. Designing better algorithms to compute the set of nodes in a node's interference region is a topic of future research.

In addition to the fact that our solution only approximates the real interference region, there are two other sources of inefficiencies in the WSA algorithm. First, the node which wins a slot may not have anything to send during that slot. To address this problem we use a simple timer mechanism, called *inactivity timer*. When k=1, each node n_i initializes an inactivity timer at the beginning of each time slot. Let n_j be the winner of that slot. If the timer of node n_i expires, and the node still has not heard any transmissions from n_j, it assumes that the slot is free. If n_i has the next highest hash value after n_j it starts transmitting in that slot. When k=2, nodes within one hop of n_j will announce to n_i that n_j is silent. To suppress multiple announcements from one-hop neighbors of n_j to n_i, we enforce that of all the nodes within one-hop of both n_i and n_j, only the node with the highest hash value will notify n_i. In practice, we set the inactivity timer to the time it takes to transmit three maximum-sized packets. This helps us avoid false-positives due to packet losses.

The second source of inefficiency is due to race conditions caused by overlapping regions. Consider three nodes n_i, n_j, n_k such that n_i and n_k do not interfere with each other, but both n_i and n_k interfere with n_j. Assume the random numbers of the three nodes are such that H_i<H_j<H_k. Then, n_i will not transmit because n_j has a higher random number, and n_j will not transmit because n_k has a higher random number. Thus, although n_i and n_k do not interfere, they would not be able to transmit in the same slot. This problem is alleviated by the technique presented in Section 4.2.4. Specifically, this technique ensures that only a small number of nodes located close to each other compete for a given slot.

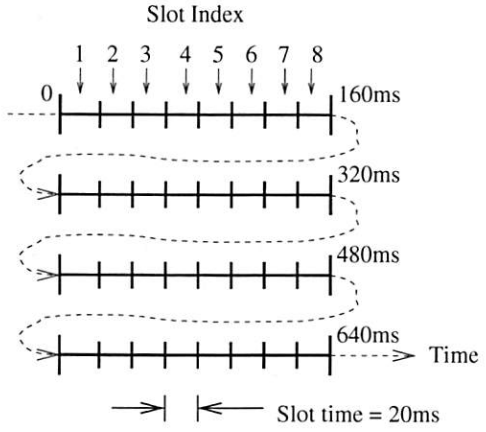


Figure 3: Groups of slots used in the overlay MAC

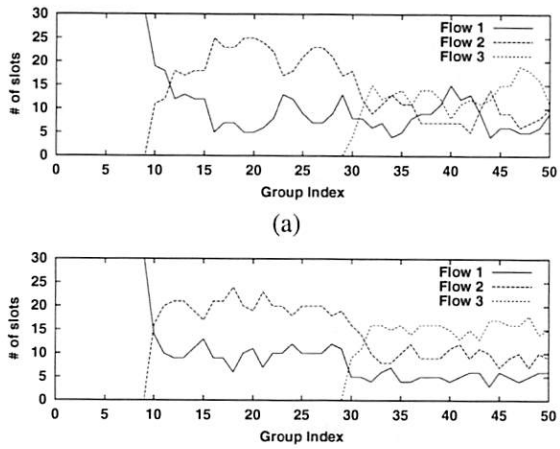


Figure 4: Number of slots in every group when using (a) default weights (w_i), and (b) inflated weights (w'_i).

4.2.4 Amortizing the cost of contention resolution

Even though our contention resolution mechanism is fairly light-weight, it is more expensive than the hardware based contention mechanisms used in 802.11, CSMA/CD, and RTS/CTS, respectively. For example, some fraction of a slot might be wasted due to the inactivity timer.

Next we present a simple mechanism to amortize the cost of the contention resolution. The idea is to form groups of N consecutive slots as shown in Figure 3. The *index* of a time-slot is defined as the position of the time-slot within a group of N slots. If a node is allowed to transmit in the slot with index i, the node is implicitly allowed to transmit in the index i of the next group with probability p. In other words, the node will relinquish slot i with probability $1 - p$. Once the node relinquishes the slot, other nodes are allowed to compete for the slot. This mechanism amortizes the cost of contention by a factor of $1/(1-p)$.

Another advantage of this mechanism is that when a node relinquishes a slot, only the nodes within its interference region are able to compete for the slot. Nodes farther away are likely to be restricted from competing since they are within range of another node that did not relinquish this particular slot. Since contention is typically restricted to nodes within a small region, the race condition described in the previous section is much less likely to occur.

However, these advantages do not come for free. A node needs now to wait for $1/(1-p)$ slots on average before it can compete. As a result it will take the system longer (*i.e.,* by a factor of $1/(1-p)$) to converge to the fair allocation when a new node joins or leaves the competition.

To address this issue, we make a slight modification to our earlier definition of of H_i. Let s_i be the number of slots owned by the node n_i when contending for a time slot. We now redefine

$$H_i = H(n_i, t)^{\frac{W s_i/N}{w_i^2}}$$

In other words, if the true weight of node n_i is w_i, node n_i uses a virtual weight $w'_i = \frac{w_i^2}{W s_i/N}$ instead of w_i. In steady state, $s_i \approx \frac{w_i N}{W}$ and hence $w'_i \approx w_i$. Thus, w'_i inflates a nodes weight when it has less than its fair share of slots, but diminishes its weight when it has more than its fair share. Thus, a new sender that becomes active will gain about $(1-p)N$ slots in each round and quickly ramp up to its fair rate.

To illustrate the advantage of using an inflated weight w'_i, we use a simple simulation involving three flows with weights 1, 2, and 3, respectively. We assume $N = 30$, and $p = 0.95$. Figure 4 plots the number of slots allocated to each flow as a function of the group index. As expected, when using inflated weights allow new flows to reach their fair allocation faster. In addition, the fluctuations in steady state behavior are smaller than in the case of using the default weights.

The choice of N impacts both the short-term fairness and the time a winner needs to wait to receive subsequent slots without competing again. At the limit, if $N = 1$, the winner is allocated $1/(1-p)$ consecutive time slots, which hurts the short-term fairness. If $N = \infty$, the winner is allocated one slot at a time as in the baseline algorithm. Based on our experimental results, we chose $N = 20$.

4.2.5 Reducing the control overhead

The basic OML algorithm employs control messages to signal (a) when a node relinquishes a slot, and (b) when

a new node enters the competition for a slot. Next, we present two simple optimizations to reduce this signaling overhead.

First, we make use of another pseudo-random function H' to decide if a node should relinquish its slot, *i.e.,* if node n_i owns the slot t, it will give up the slot only if $H'(n_i, t) < (1 - p)$. Again, H' is a pseudo-random function with range $(0, 1]$ and other nodes can compute this value without any communication.

Second, we include the queue length at the sender in the header of each packet. Thus, if the queue becomes empty during the current slot, the node's neighbors are implicitly notified, and the remaining node with the next highest H_i is allowed to transmit for the rest of the slot. If on the other hand the queue of the sender is not empty at the end of the time slot, then its one-hop neighbors will implicitly assume that the node will compete in the next time slot.

In summary, only nodes that join the competition need to transmit individual control messages. All the additional information about active nodes in the interference region is piggy-backed in the packet header. To deal with node failures, we remove a node from the list of contenders if we do not receive an update on its queue-length for an extended period of time.

4.3 Putting everything together

Figure 5 shows the WSA pseudocode. For readability, here we assume that the interference region is the same as the transmission region, *i.e.,* $k = 1$.

Each node maintains a list of all nodes in its interference region that are active, and a map of which time slot is allocated to which node. Since nodes are in promiscuous mode, each node can maintain the list of active nodes by simply inspecting the queue lengths in the headers of the packets it receives. If a node does not hear any message from a neighbor within a predefined interval of time, it assumes that the neighbor is no longer competing and removes it from its list. This allows the algorithm to be robust in the presence of packet losses and node failures.

To implement a two hop interference region, a node simply piggybacks the information about all its one-hop neighbors in the packets it sends. This information is spread across the headers of all packets it sends during the slot to reduce the per-packet overhead. This allows each node to learn about its two-hop neighborhood. Similarly, when a node becomes active, it broadcasts a control message to its two-hop neighborhood.

```
function scheduler()
  while (TRUE)
    getSlotIndex(&slotId, &slotIndex);
    // check whether you already own the slot
    if (owner[slotIndex] ! = myAddress)
      setInactivityTimer();
    if (H'(slotId, owner[slotIndex]) < (1 − p))
      // contendForSlot updates owner[ ]
      contendForSlot(slotId, slotIndex);
    while (!endOfCurrentSlot())
      if (owner[slotIndex] == myAddress)
        p = getPacket(omlQueue);
        if (p) send(p);

function recvPacket(p)
  if (p.type == CONTEND or p.header.q_size > 0)
    active_list.add(p.src);
    // don't content for this slot
    cancelInactivityTimer();
    // active list may have changed; recompute winner
    computeWinner(slotId, slotIndex);
  if (p.type == RELINQUISH or p.header.q_size == 0)
    contendForSlot(slotId, slotIndex);
  if (p.header.q_size == 0)
    active_list.remove(p.src);
  if (p.type == DATA)
    // deliver packet locally if this node is
    // the destination; otherwise route it
    processData(p);
function contendForSlot(slotId, slotIndex)
  if (!isEmpty(omlQueue) and totalOwnedSlots == 0)
    // other nodes may not be aware this node is active
    sendContentionMessage();
  computeWinner(slotId, slotIdx)

function handleInactivityTimer()
  contendForSlot(slotId, slotIndex);

function cleanActiveList()
  // This function is called periodically by every node
  // to time-out failed nodes from the active_list
  for all n ∈ active_list
    if (curr_time - n.time_added > FAIL_TIMEOUT)
      active_list.remove(n);
```

Figure 5: The WSA algorithm for an one-hop interference region.

5 Experimental Test-bed

In order to motivate our work in Section 3, we use some examples of actual observed performance in an 802.11 testbed. In this Section, we describe the testbed that we use for these results. Our testbed currently consists of 6 wireless nodes based on commodity hardware and software.

The hardware consists of small form-factor computers equipped with a 2.4GHz Celeron processor and 256MB of RAM. Each computer is also equipped with a Netgear WAG511 [6] tri-mode PCMCIA wireless Ethernet adapter. This card is capable of operating in 802.11a,b and g modes, but we conduct all our experiments in 802.11a mode to avoid interference with another production 2.4GHz wireless network operating in same environment.

We have installed Linux (kernel 2.4.22) in all these systems along with the MadWiFi [3] driver for the wire-

less cards. For routing, we use the Click software router [26] because it provides an easily extensible modular framework. Also, the MIT Grid [4] project provides a readily download-able implementation of DSR [25] and AODV [29] on top of Click. More details about our implementation of OML are available in [32].

6 Simulation Results

In this Section, we evaluate the benefits of OML by using Qualnet [7], a commercial packet-level wireless simulator. Our results can be summarized as follows. First, despite the additional control overhead, the throughput of an OML/WSA network is comparable to the throughput of an 802.11 network. This is because the loss in throughput due to the control overhead is offset by the fact that OML/WSA experiences much lower contention. Second, WSA significantly improves the fairness in a multi-hop network. In particular, while in a multi-hop 802.11 network, a significant percentage of TCP flows are shut-off, OML/WSA completely avoids this problem.

To quantify the fairness, we use the metric defined by Chiu and Jain in [14]. Consider a system with M flows, with weights $w_1, w_2, \ldots, w_M$, each receiving a throughput $x_1, x_2, \ldots, x_M$. The fairness index F is defined as

$$F = \frac{(\sum_i x_i/w_i)^2}{M \sum_i x_i^2/w_i^2}$$

Note that $F = 1$ when each flow's throughput is exactly in proportion to its weight, and $F = 1/M$ when only one flow receives the entire throughput.

We next describe the simulation setting. Each node in the simulation is equipped with an 802.11a network interface operating at 6 Mbps. We use the outdoor two-ray propagation model which yields a radio range of about 350 m. In all experiments, we maintain the density of the network constant at 50 nodes per square km, and place the nodes at random locations. The time it takes to transmit an 1500 byte packet, including the overhead of the MAC and physical layers is about 2 ms. Hence, we choose a slot time of $l = 10$ ms, and a group with $N = 20$ slots. We have disabled the RTS-CTS handshake in all simulations since this maximizes the performance of both the baseline 802.11 and the OML networks. We use the AODV [29] routing algorithm, and run each simulation for one minute (simulation time).

6.1 Collisions and Throughput

As mentioned in Section 4, OML/WSA can achieve better fairness than native 802.11 in a multi-hop network. To evaluate the effects of this trade-off on throughput, in this experiment we vary the size of the network from 15 to 50 nodes, and measure the throughput achieved by a number of simultaneous UDP flows. All flows originate at different nodes, but have the same sink. Such a traffic pattern models a multi-hop network used to access the Internet through a single gateway.

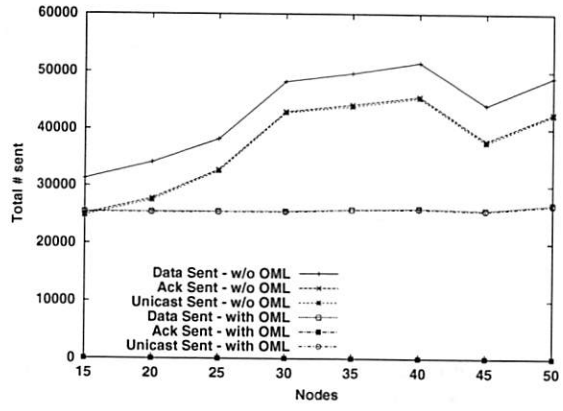


Figure 6: Packet retransmissions in a multi-hop network with 10 flows

Figure 6 shows the effect of the size of the network on the total number of (a) DATA packets and (b) ACKs successfully transmitted at the data link layer, and the (c) unicast packets successfully delivered to the destination. For each network size we consider 10 simultaneous UDP flows. The difference between (a) and (c) gives the total number of packet retransmissions at the MAC layer. Of these (a)-(b) are due to the loss of DATA packets, and (b)-(c) are due to the loss of ACK packets. Because in these simulations OML makes the conservative assumption that all nodes in a two-hop neighborhood interfere, there is less frequency reuse and the total number of transmissions attempted is lower than in 802.11. On the other hand, since OML does not allow competing senders to be active at the same time, there is hardly any MAC layer retransmission. In contrast, in the case of 802.11, up to 25% of the packets are retransmitted. Note that applications that use broadcast transmissions (*e.g.*, routing protocols) can greatly benefit from fewer collisions since broadcast packets are not acknowledged and retransmitted at the MAC layer.

Figure 7 plots the average throughput of 5 or 10 simultaneous flows both with and without OML. Even though OML uses additional control overhead and permits less frequency reuse, the throughput with OML is close to the throughput without OML. Also, there is fundamental trade-off between throughput and fairness in multi-hop networks [22] since longer flows consume more resources than 1-hop flows. But by avoiding collisions and using the medium more efficiently, OML provides much better fairness (see Section 6.2) without sacrificing the throughput.

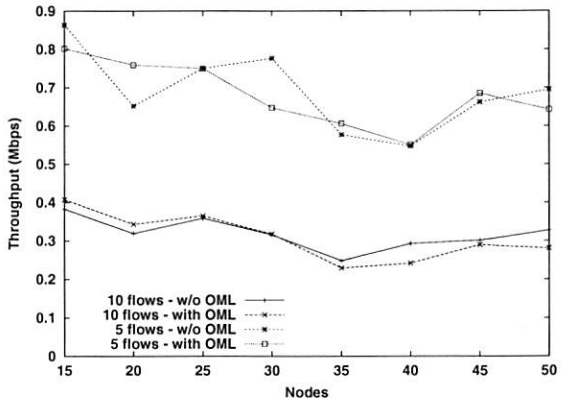


Figure 7: Average throughput in a multi-hop network with and without OML

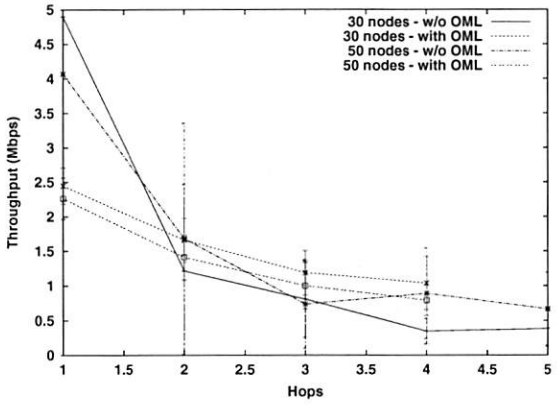


Figure 9: Relationship between throughput and the number of hops

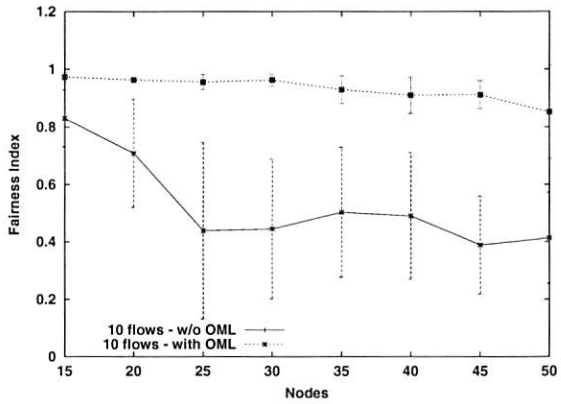


Figure 8: Fairness index of 10 simultaneous flows

6.2 Fairness

A well-known problem with multi-hop 802.11 networks is that short flows, in terms of the number of hops they traverse, receive a much higher throughput than long flows. To show how OML/WSA can alleviate this problem, in the following experiment we set the weight of each node to be equal to the number of unique IP source addresses seen in its output queue; this roughly approximates having unit weight for each flow since there is only one flow starting from each node. Thus a node that forwards packets from m flows will have weight m. Within a single node, we maintain separate queues for each IP source address and implement round-robin scheduling between these queues. If all nodes contend with each other, the weight allocations and the scheduling algorithm will ensure that all flows receive equal throughput irrespective of their length.

Figure 8 shows the fairness index of 10 simultaneous flows averaged over 10 simulation runs. This value can vary from 0.1 (least fair) to 1 most fair in this case. The fairness index in the case of 802.11 is mostly below 0.5 for large networks. This is because in a large network there are more flows with longer routes. In contrast, the fairness index in the case of OML/WSA is above 0.8 even for a 50 node network. The reason the fairness is not 1 is simply because because not all flows compete with all other flows. The only way to achieve $F = 1$ in multi-hop network is to constrain all flows to the throughput of the flow experiencing the most contention, which leads to inefficient use of the channel in the less congested areas.

Figure 9 plots the average throughput of the flows as a function of their length. As expected, 802.11 favors shorter flows, but OML/WSA is more fair. For example, in the case of the 30-node network, the average throughput of four-hop flows under 802.11 is less than 10% of the throughput of one-hop flows. In contrast, with OML, the average throughput of four-hop flows is more than half the throughput of one-hop flows.

7 Results from the testbed

In this section, we evaluate OML/WSA using our experimental testbed. Our main results can be summarized as follows. Section 7.1 shows that OML/WSA can increase the overall throughput of the system by fairly allocating transmission time, instead of transmission opportunities like the 802.11 protocol. Section 7.2 shows that even in a simple chain topology involving one-hop flows, 802.11 can cause a significant number of TCP flows to experience starvation, while OML avoids this phenomenon. Section 7.3 shows that, as expected, the starvation problem is even worse in the case of multi-hop routing, but OML can still handle this problem. In Section 7.4, we evaluate the allocation accuracy of WSA in a simple scenario.

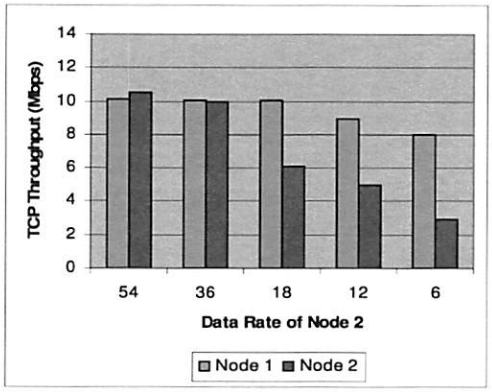


Figure 10: 802.11 throughput in the presence of heterogeneous data rate senders using OML

7.1 Heterogeneous data rates

In this experiment, we reconsider the scenario described in Section 3.1.2 where two nodes are simultaneously sending one flow each to the access point (unlike the rest of this section, we conduct this experiment in infrastructure mode). One node operates at 54 Mbps, and the other node operates at rates ranging from 6 to 54 Mbps. We use the WSA algorithm to allocate the bandwidth, with each node having the same weight. This leads to each node receiving an equal channel-access time, rather than an equal number of transmission opportunities as in 802.11. Note that this allocation implements the temporal-sharing policy as proposed in [34]. Figure 10 shows the throughputs of both flows. The two flows receive throughputs approximately proportional to the rates they are operating at, and the total system throughput drops less than in the case of 802.11 when the second node is operating at 6 Mbps.

7.2 Chain topology

In this experiment, we configure our testbed as a five-hop chain, and pick two random links in this chain. Figure 11 shows the CDF of the flows' throughputs along these links over 50 trials. We consider four scenarios: (a) the flows are started one at a time (b) the flows are started at the same time (*i.e.,* simultaneous flows) using baseline 802.11 (without OML), (c) simultaneous flows using OML assuming one-hop interference regions ($k = 1$), and (d) simultaneous flows using OML assuming two-hop interference regions ($k = 2$). When using OML each flow is assigned a unit weight.

As expected, in scenario (a) when only one flow is active, the flows get very good throughput. In 96% of the cases the throughput is greater than 4Mbps. However, in scenario (b) when the flows are simultaneously active without OML, in 24% of the cases, one of the two flows is not even able to establish a TCP connection. This is shown in Figure 11, where 12% of all flows have zero throughput. Furthermore, about 20% of all flows receive less than 1.5 Mbps. In contrast, when using OML with $k = 2$ only about 10% of the flows have a throughput below 1 Mbps. The results when using OML with $k = 1$ are not as good: we only eliminate 6% of the starvation cases. On the other hand, more flows achieve a higher throughput when $k = 1$: around 35% of all flows have throughputs around 4.5 Mbps. These results illustrate the trade-off between using one-hop and two-hop interference regions. Using a two-hop interference region leads to more conservative spatial reuse of the channel. Hence, there are cases in which two nodes cannot transmit simultaneously even though they do not interfere with each other. In contrast, using a one-hop interference region leads to higher throughput, but at the cost of hurting the fairness.

	OML $k = 1$	OML $k = 2$	OML Oracle	No OML
Throughput	5.7742	5.1784	5.5712	5.8654
Fairness Index	0.821	0.913	0.922	0.803

Table 1: Average system throughput (Mbps) and fairness for two simultaneous flows

Table 1 further illustrates the trade-off between throughput and fairness as determined by the value of k. The table shows the average aggregate throughput (*i.e.,* the sum of throughput of both flows) and the average fairness index, F, over 50 trials. In addition to the previous experiment scenarios, we consider another one, called "OML with Oracle". This scenario is intended to show the best results OML can achieve when it has complete information about the interference pattern. To implement this scenario, we measure the interference between any two flows in advance and pre-load this information into each node. Recall that for $N = 2$, F ranges from 0.5 (least fair) to 1.0 (most fair). Not surprisingly, in the oracle scenario we obtain the best fairness, without significantly sacrificing the throughput. OML with $k = 1$ comes close to the oracle in terms of throughput, whereas OML with $k = 2$ comes close in terms of fairness. Finally, note that native 802.11 achieves the highest throughput, but the lowest fairness. This is because when one link has a better signal quality than the other, it allows the better link to transmit all the time at the expense of starving the weaker link.

7.3 Multi-hop routing

In this section, we study how OML can improve the flow throughputs in a multi-hop routing network. For this pur-

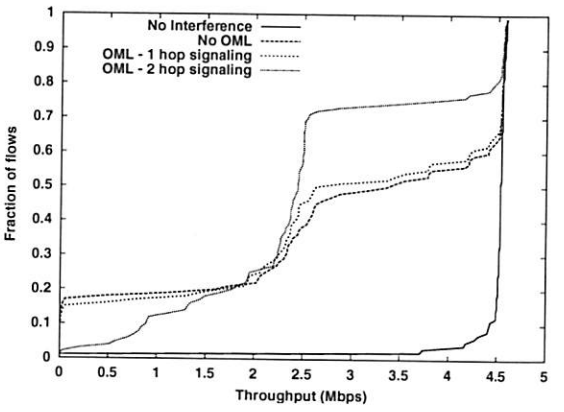


Figure 11: CDF of throughput of 1-hop flows in the network

pose, we arrange the nodes in a simple Y-shaped topology and initiate three TCP flows. Flows A and B originate at nodes 1 and 2, respectively, and terminate at node 4, after being routed via node 3. Flow C is an one-hop flow from node 5 to node 4. The weight of each node is proportional to the number of flows it sees: the weights of nodes 1, 2, and 5 are one, the weight of node 3 is two, and the weight of node 4 is three. In this experiment, we use OML with two-hop interference regions.

First we note that when only one of the three flows is active, flows A and B receive about 2.24 Mbps, while flow C receives 4.55 Mbps, with or without OML.

Table 2 shows the throughputs of each flow when more than one flow is active. Without OML, flow C effectively shuts-off the two-hop flows. In contrast, OML allocates at least 1 Mbps to each flow, when only one of the two-hop flows is active, and at least 0.94 Mbps, when all flows are active. The reason why the throughputs of the two-hop flows are not higher is because the two-hop interference assumption is violated. According to this assumption, transmissions from nodes 1 or 2 should not interfere with with transmissions from node 5. However, this assumption did not hold in our experiment. Let flow D be the one-hop flow from node 1 to 3. When flows C and D were simultaneously active, C received 4.03 Mbps whereas D received only 0.89 Mbps. This shows that transmissions from nodes 1 and 5 do indeed interfere.

7.4 Weighted allocation

In this experiment, we evaluate the accuracy of WSA, and thus its ability to provide per-node service differentiation by setting the node weights. We consider two nodes A and B that send a TCP flow each to a third node C. We set the weight of node A to 1, and assign a weight ranging from 1 to 4 to node B. Table 3 shows the throughput achieved by the nodes for each combination of weights.

Active Flows	Throughput (Mbps)		
	Flow A	Flow B	Flow C
A&C w/o OML	0.14		4.40
A&C with OML	1.69		2.36
B&C w/o OML		0.06	4.48
B&C with OML		1.71	2.08
A&B w/o OML	1.31	1.65	
A&B with OML	1.11	1.11	
A,B&C w/o OML	0.11	0.03	4.42
A,B&C with OML	0.96	0.94	1.96

Table 2: Interaction of flows in an ad-hoc topology

Weight of A	1	1	1	1
Throughput of A	2.26	1.48	1.36	0.92
Weight of B	1	2	3	4
Throughput of B	2.20	2.97	3.06	3.64

Table 3: Throughput received by senders with different weights

As expected, the ratio of throughputs obtained by the two nodes tracks the ratio of their weights closely.

8 Open Issues and Limitations

The current design of OML should be viewed as a first iteration. As we gather more experience with using OML, we expect the OML design to evolve significantly.

Though we believe that the current design of OML does not require any changes to handle *mobility*, since OML relies on neighborhood information being propagated quickly, we do expect a performance penalty as mobility increases. Quantifying this overhead, and further optimizing OML if needed, remains an open problem.

OML does not work well in the presence of interference from native 802.11 clients. In this case, *all* OML nodes will receive an aggregate bandwidth equivalent to that of a *single* 802.11 node. A possible solution would be to detect the presence of competing traffic and allow multiple OML nodes to send data in the same time slot. Finally, in multi-hop networks the interplay between the flow constraints and the interference constraints makes it difficult to precisely formalize the notion of fairness achieved by WSA. We plan on further studying how to define and achieve weighted fairness in a multi-hop network.

9 Conclusions

In this paper, we have described the design and the implementation of an overlay MAC layer (OML) solution which addresses some of the limitations of the 802.11

MAC layer. By using both simulation and experimental evaluation, we show that while 802.11 may cause TCP flows to experience starvation, OML can avoid this problem. In addition, we show that OML can reduce the contention in the network, and provide service differentiation among nodes, with relatively low control overhead.

The power of the OML approach is that it allows us to implement MAC layer functionality *without* modifying the existing 802.11 protocol. In this respect, our approach is reminiscent of the overlay network solutions that aim to implement network layer functionality such as resilient routing on top of the existing IP layer. Ultimately, OML would enable us to experiment with new scheduling and bandwidth management algorithms, and evaluate their benefits to the existing applications, before implementing these algorithms in the MAC layer.

References

[1] AiroPeek NX - Wireless Network Protocol Analyzer. http://www.wildpackets.com/.

[2] Ethereal - Network Protocol Analyzer. http://www.ethereal.com/.

[3] MadWifi. http://madwifi.sourceforge.net/.

[4] MIT grid project. http://www.pods.lcs.mit.edu/grid/.

[5] MIT roofnet. http://www.pdos.lcs.mit.edu/roofnet/.

[6] Netgear. http://www.netgear.com/.

[7] The Qualnet Simulator from Scalable Networks Inc. http://www.scalable-networks.com/.

[8] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient overlay networks. In *SOSP*, 2001.

[9] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *SIGCOMM*, 2002.

[10] L. Bao and J. Garcia-Luna-Aceves. Hybrid channel access scheduling in ad hoc networks. In *ICNP*, 2002.

[11] L. Bao and J. J. Garcia-Luna-Aceves. Distributed dynamic channel access scheduling for ad hoc networks. In *Journal of Parallel and Distributed Computing*, 2002.

[12] V. Bharghavan, S. Lu, and T. Nandagopal. Fair queueing in wireless networks: Issues and approaches. *IEEE Personal Communications Magazine*, 1999.

[13] S. Biswas and R. Morris. Opportunistic routing in multi-hop wireless networks. In *HOTNETS*, 2003.

[14] D.-M. Chiu and R. Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Comput. Netw. ISDN Syst.*, 17(1):1–14, 1989.

[15] Y.-H. Chu, S. G. Rao, and H. Zhang. A case for end system multicast. In *SIGMETRICS*, pages 1–12, Santa Clara, CA, 2000.

[16] I. Cidon and M. Sidi. Distributed assignment algorithms for multihop packet radio networks. *IEEE Trans. Comput.*, 38(10):1353–1361, 1989.

[17] Wireless networking reference - community wiress/rooftop systems. http://www.practicallynetworked.com/.

[18] D. D. Couto, D. Aguayo, J. Bicket, and R. Morris. A high-throughput path metric for multi-hop wireless routing. In *MOBICOM*, 2003.

[19] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *SIGCOMM*, 1989.

[20] R. Draves, J. Padhye, and B. Zill. Comparison of routing metrics for multi-hop wireless networks. In *SIGCOMM*, 2004.

[21] J. Elson and D. Estrin. Time synchronization for wireless sensor networks. In *PDPS Workshop on Parallel and Distributed Computing Issues in Wireless Networks and Mobile Computing*, pages 186–186.

[22] V. Gambiroza, B. Sadeghi, and E. Knightly. End-to-end performance and fairness in multihop wireless backhaul networks. In *MOBICOM*, 2004.

[23] M. Heusse, F. Rousseau, G. Berger-Sabbatel, and A. Duda. Performance anomaly of 802.11b. In *INFOCOM*, San Francisco, USA, March-April 2003.

[24] K. Jain, J. Padhye, V. Padmanabhan, and L. Qiu. Impact of interference on multi-hop wireless network performance. In *MOBICOM*, 2003.

[25] D. B. Johnson, D. A. Maltz, and J. Broch. DSR: The Dynamic Source Routing protocol for multihop wireless ad hoc networks. In *Ad Hoc Networking*, 2001.

[26] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, 2000.

[27] L. M. S. C. of the IEEE Computer Society. Wireless LAN medium access control (MAC) and physical layer (PHY) specifications. *IEEE Standard 802.11*, 1999.

[28] A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM Trans. Netw.*, 1(3):344–357, 1993.

[29] C. Perkins. Ad hoc On demand Distance Vector (AODV) routing. *IETF Internet Draft*, 1997.

[30] L. Pond and V. Li. A distributed time-slot assignment protocol for mobile multi-hop broadcast packet radio networks. In *MILCOM*, 1999.

[31] B. Raman and K. Chebrolu. Revisiting MAC design for an 802.11-based mesh network. In *HOTNETS*, 2004.

[32] A. Rao and I. Stoica. An overlay MAC layer for 802.11 networks. Technical Report UCB//CSD-04-1317, University of California, Berkeley, 2004.

[33] K. Romer. Time synchronization in ad hoc networks. In *Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing*, pages 173–182. ACM Press, 2001.

[34] B. Sadeghi, V. Kanodia, A. Sabharwal, and E. Knightly. Opportunistic media access for multirate ad hoc networks. In *MOBICOM*, 2002.

[35] U. Schmid and K. Schossmaier. Interval-based clock synchronization. *Real-Time Systems*, 12(2):173–228, 1997.

[36] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirection infrastructure. In *SIGCOMM*, 2002.

[37] C. A. Waldspurger and W. E. Weihl. Lottery scheduling: Flexible proportional-share resource management. In *Operating Systems Design and Implementation*, pages 1–11, 1994.

[38] Y. Yi and S. Shakkottai. Hop-by-hop congestion control over a wireless multi-hop network. In *INFOCOM*, 2004.

Notes

[1] NAV is maintained internally in the hardware to keep track of RTS, CTS and reservations in the packet header.

[2] We use the term interference region to refer to a set of nodes that compete with each other for access to the channel.

DESIGN AND IMPLEMENTATION OF A SINGLE SYSTEM IMAGE OPERATING SYSTEM FOR AD HOC NETWORKS

Hongzhou Liu Tom Roeder Kevin Walsh Rimon Barr Emin Gün Sirer
Department of Computer Science, Cornell University
liuhz,tmroeder,kwalsh,barr,egs@cs.cornell.edu

ABSTRACT

In this paper, we describe the design and implementation of a distributed operating system for ad hoc networks. Our system simplifies the programming of ad hoc networks and extends total system lifetime by making the entire network appear as a single virtual machine. It automatically and transparently partitions applications into components and dynamically finds them a placement on nodes within the network to reduce energy consumption and to increase system longevity. This paper describes our programming model, outlines the design and implementation of our system and examines the energy efficiency of our approach through extensive simulations as well as validation of a deployment on a physical testbed. We evaluate practical, power-aware, general-purpose algorithms for component placement and migration, and demonstrate that they can significantly increase system longevity by effectively distributing energy consumption and avoiding hotspots.

1 Introduction

Ad hoc networks simultaneously promise a radically new class of applications and pose significant challenges for application development. Recent advances in low-power, high-performance processors and medium to high-speed wireless networking have enabled new applications for ad hoc and sensor networks, ranging from large-scale environmental data collection to coordinated battlefront and disaster-relief operations. Ad hoc networking applications differ from traditional applications in three fundamental ways. First, ad hoc networking applications are inherently distributed. Operating on a distributed platform requires mechanisms for remote communication, naming, and migration. Second, ad hoc networks are typically highly dynamic and resource-limited. Key performance metrics, such as bandwidth, may vary through several orders of magnitude, and mobile nodes are typically limited in energy. Consequently, applications need policies for using available resources efficiently, and sharing them among competing applications fairly. Finally, ad hoc networking applications are expected to outlast the lifetime of any one node. Performing long-running computations in a dynamic environment requires facilities for dynamically introducing new functionality and integrating it with existing computations present in the network. Current operating systems, however, provide little support for ad hoc networks.

The current state of the art in developing applications for ad hoc networks is to treat the network as a system of standalone systems, that is, a network comprised of independent, autonomous computers. This programming model forces applications to provide all of their requisite mechanisms and policies for their operation themselves. Mechanisms, such as those for distributing code and migrating state, as well as policies, such as how to react to diminishing battery supply on a given node, need to then be embedded, independently, in all applications. Such a limited programming model not only makes developing ad hoc networking applications tedious and error-prone, but the lack of a global operating system acting as a trusted arbiter between mutually distrusting applications allows inter-application conflicts to emerge. Critical global properties of the network, such as system longevity, are dictated by distributed policies encoded in applications; network operators have little control over the operation of their systems, as there is no network-wide system layer. This situation is analogous to the early standalone operating systems implemented entirely in user-level libraries, in that assuring global properties of the system requires whole system analysis, including auditing all application code.

In this paper, we investigate an alternative programming model for ad hoc networks where a thin distributed operating system layer makes the entire network appear to applications as a single virtual machine. We describe the design and implementation of a distributed operating system based on this model, called MagnetOS. We show that the high level of abstraction presented to applications with such a model not only simplifies the development of applications substantially, but also enables the underlying system to make energy-efficient placement and migration decisions. Unlike distributed programming on the Internet, where energy is not a constraint, delay is low, and bandwidth is plentiful, physical limitations of ad hoc networks lead to some unique requirements. Technology trends indicate that the primary limitation of mobile ad hoc and sensor networks is energy consumption, and communication is the primary energy consumer [38]. Consequently, the goals of MagnetOS are as follows:

Efficiency: The system should execute distributed ad hoc network applications in a manner that conserves power and extends system lifetime. Policies and mechanisms used for adaptation in the systems layer should not require excessive communication or power consumption.

Adaptation: The system should respond automatically to significant changes in network topology, resource availability, and the communication pattern of the applications. Adaptation should not require a priori involvement from the application programmer.

Generality: The system should support a wide range of applications. Developing new applications that execute efficiently on an ad hoc network should require little effort. The system should provide effective default adaptation policies for applications that are not power-aware. Applications should be able to direct and, when executed with sufficient privilege, override the default adaptation using application-specific information.

Extensibility: The system should provide facilities for deploying, managing and modifying executing applications whose lifetime may exceed those of the network participants.

Compatibility: The system should not require mastering a new paradigm in order to deploy applications. Standard development tools should continue to work in building applications for ad hoc networks. The system should enable applications to execute on ad hoc networks of heterogeneous nodes.

MagnetOS meets these goals by making the entire network operate as an extended Java virtual machine. MagnetOS applications are structured as a set of interconnected, mobile event handlers, specified statically by the programmer as objects in an object-oriented system. The MagnetOS runtime uses application partitioning and dynamic migration to distribute the event handlers to nodes in the ad hoc network and finds an energy-efficient placement of handlers. MagnetOS applications are comprised of event handlers that communicate with each other by raising well-typed events. Event signatures specify the types of the arguments passed with the event, as well as the return type of the handler. By default, all externally visible entry points, such as methods in a Java object specification, serve as event declarations, and method bodies constitute the default handler for that event in the absence of overriding runtime event bindings. Consequently, the MagnetOS programming model closely parallels the Java virtual machine, providing access to standard Java libraries and enabling familiar development tools to be used to construct distributed applications.

Our MagnetOS implementation consists of a static application partitioning service that resides on border hosts capable of injecting new code into the network, a runtime on each node that performs dynamic monitoring and component migration and a set of policies to guide object placement at run time. The static partitioning service takes regular Java applications and converts them into distributed components that communicate via events by rewriting them at the bytecode level (Figure 1). The code injector then finds a suitable initial layout of these components and starts the execution of the application. The runtime monitors the performance of the application and migrates application components when doing so would benefit the system.

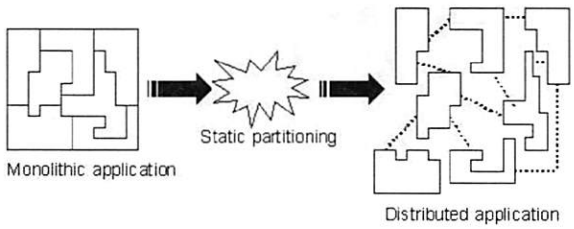


Figure 1: A static partitioning service converts monolithic Java applications into distributed applications that can run on an ad hoc network and transparently communicate by raising events.

The algorithms for event handler placement form the core of our system. We present practical, online algorithms for finding an energy-efficient distribution of application components in an ad hoc network (Figure 2). This paper examines the effectiveness of these algorithms in reducing energy consumption and extending system lifetime in the context of three application benchmarks, and examine their impact on system longevity. These algorithms operate by dividing time into epochs, monitoring the communication pattern of the application components within each epoch, and migrating components at the end of the epoch when doing so would result in more efficient power utilization.

We have implemented the system described in this paper and deployed it on x86 laptops, Transmeta tablets, and StrongArm PocketPC devices. Since current Java virtual machines place significant minimum memory requirements from their platforms, we have developed our own space-optimized JVM, suitable for "headless PocketPCs" which are cheap, have low energy requirements, and approximate what one might find in an embedded commodity device a few years from now. We report on our experience with developing and deploying applications in a physical testbed of 16 mobile nodes using this implementation. To increase the scale of our evaluation, we report results from simulation studies, validated against our implementation, which show that the MagnetOS system can achieve significant improvement in system longevity over static placement and standard load-balancing techniques.

This paper makes three contributions. It proposes a novel programming model for ad hoc networks where the entire network appears as a single unified system to the

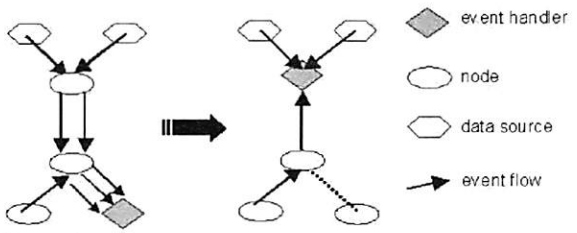


Figure 2: Migrating components closer to their data sources increases system longevity and decreases power consumption by reducing total network communication cost.

programmer. It describes the design and implementation of an operating system based on this model, where the distributed application is expressed at the language level as a single, cohesive application. The system operates by automatically partitioning applications and transparently migrating event handlers at runtime. Second, we propose practical, adaptive, online algorithms for finding an energy-efficient placement of application components in an ad hoc network. These algorithms form the central challenge to automatic energy-efficient execution in the system layer, and we demonstrate that limited, locally-collected information can lead to effective placement decisions. Finally, we demonstrate that these algorithms achieve high-energy utilization, extract low overhead, and improve system longevity.

In the next section, we describe related work on operating system support for ad hoc networks and their applications. Section 3 outlines our system implementation, including the code partitioning and distribution technique. Section 4 presents our network and application model, describes our simulation framework and evaluates within this environment. We summarize our contributions in Section 5.

2 Related Work

Past work has examined distributed operating systems, ad hoc networks, power management and programmability of ad hoc sensor network, though few systems have examined all of them.

2.1 Distributed Systems

Data and code migration have been examined extensively in the context of wired networks of workstations. Early landmark systems, such as V [9], Sprite [35], Accent [39], and LOCUS [37], implemented native operating system facilities for migrating processes between nodes on a tightly coupled cluster. Glunix [14] provides facilities for managing applications on a tightly connected networks of workstations. More recently, the cJVM [4] and JESSICA [32] projects have examined how to extend a Java virtual machine across a high-performance cluster. Others, including Condor [30] and CoCheck [43], provide user-level mechanisms for checkpointing and process migration without operating system support. These projects target high-performance, well-connected clusters. Their main goals are to balance load and achieve high performance in a local area network for interactive desktop programs or CPU-intensive batch jobs. In contrast, MagnetOS targets wireless multihop networks, where utilizing power effectively and maximizing system longevity is more important than traditional application performance.

Distributed object systems have examined how to support distributed computations in the wide area. Emerald [24] provides transparent code migration for programs written in the Emerald language, where the migration is directed by source-level programmer annotations. Thor [29] provides persistent objects in a language-independent framework. It enables caching, replication and migration of objects stored at object repositories. These seminal systems differ fundamentally from MagnetOS in that they require explicit programmer control to trigger migration, do not support an ad hoc network model and target traditional applications.

The closest approach to ours are some recent systems that focused on how to partition applications within a conventional wired network. The Coign system [22] has examined how to partition COM applications between two tightly interconnected hosts within a local-area network. Coign performs static spatial partitioning of desktop applications via a two-way minimum cut based on summary application profiles collected on previous runs. The ABACUS system [1] has examined how to migrate functionality in a storage cluster. MagnetOS shares the same insight as Coign, in that it also focuses on the automatic relocation of application components, but differs in that it dynamically moves application components in response to changes in the network, instead of computing a static partitioning from a profile. Kremer et al. [25] propose using static analysis to select tasks that can be executed remotely to save energy. J-Orchestra [46] performs application partitioning via rewriting, leaving dynamic migration decisions under application control. Spectra [13] monitors resource consumption, collects resource usage histories and uses quality of service (fidelity) information supplied by the application to make resource allocation decisions. Spectra is invoked prior to operation startup, and statically determines a location at which to execute the operation.

Middleware projects have looked at constructing toolkits to support mobile applications. The Rover toolkit [23] provides relocation and messaging services to facilitate the construction of mobile applications. The Mobiware [3] and DOMT [26] toolkits are targeted specifically for ad hoc networks and provide an adaptive QoS programming interface. XMIDDLE [52] assists with data management and synchronization. MagnetOS takes a systems approach instead of providing a program-

mer driven toolkit and automatically manages the shared network and energy resources among ad hoc network applications. This approach unifies the system layer and ensures that disparate applications, regardless of which toolkits they use, behave in a cooperative manner.

2.2 Ad hoc Routing Protocols

There has been much prior research on ad hoc routing algorithms. Proactive, reactive and hybrid routing protocols seek to pick efficient routes by proactively disseminating or reactively discovering route information, or both. While some protocols, such as PARO [16] and MBLR [47], have examined how to make power-aware routing decisions, all of these routing algorithms assume that the communication end-points are fixed. Directed diffusion [19] provides a data-centric programming model for sensor networks by labeling sensor data using attribute-value pairs and routing based on a gradient. MagnetOS complements the routing layer to move application code around the network, changing the location of the communication endpoints and radically altering the communication pattern of the overall application. It provides increased system and application longevity by bringing application components closer to the data sources, which complements the route selection performed by the ad hoc routing protocol.

2.3 Operating Systems

Prior work has examined how to construct space-constrained operating systems for sensor networks. TinyOS provides essential OS services for sensor nodes with limited hardware protection and small amounts of RAM [20]. Maté [27] provides a capsule-based programming model for in-network processing on sensor nodes. Impala [31], the middleware layer of ZebraNet, introduces a system activity model that handles a mix of regular operations and irregular events for long-running mobile sensor systems. MagnetOS is complementary to these standalone systems, in that its system-wide abstractions can be built on top of the services they provide.

2.4 Programming Paradigms for Wireless Networks

There have been many research efforts to improve the programmability of wireless ad hoc and sensor networks. Hood [51] and Abstract Regions [50] simplify application development by providing high level abstractions which group together nodes with similar properties. Hood provides a low-level data sharing mechanism among neighboring nodes. Abstract Regions is similar to Hood but also provides data aggregation and resource tuning options. TAG [33] treats the sensor network as a database and performs data aggregation. Sensorware [6] proposes an active sensor framework which employs lightweight and mobile control scripts. By contrast, MagnetOS takes a system approach that makes the whole network appear as a single system image to applications and does not require programmers to master a new programming paradigm.

2.5 Power Efficiency

Previous work has also examined how to minimize power consumption within an independent host through various mechanisms ([36], [17], [11], [44], [7], [41]), including low-power processor modes, disk spin-down policies, adapting wireless transmission strength and selectively turning off unused devices. More recent work [2] improves the power management by exposing new interfaces to applications and devices. Our system is complementary to this work and opens up further opportunities for minimizing power consumption by shipping computation out of hosts limited in power to less critical nodes.

Remote execution, i.e. moving costly computation to well-equipped nodes, has been used ([5], [18], [28]) to conserve the energy of mobile devices. Others ([8], [15]) have proposed improving energy efficiency by controlling node mobility. Some of this work shares the same insight as MagnetOS in that they also try to improve power efficiency by application partitioning and computation migration. However, MagnetOS does not require the aid of well-equipped nodes or the assumption that node mobility is controllable.

3 System Overview

MagnetOS provides a single system image programming model based on events. An event is an indication that a named piece of code should be executed in response to a system, sensor, or application-initiated occurrence. Events are typed and may carry arguments and return values; they are defined in MagnetOS as procedures. Events may be synchronous, in which case the event invoker blocks until termination of the handler, or asynchronous, in which case control returns immediately to the component that raised the event. Applications consist of a set of event handlers, and execution consists of a set of event invocations that may be performed concurrently. Every application receives an initial event to start its execution; as well, components of an application receive specific events for initialization. Device drivers, sensors, and other peripherals may define events suitable for applications to handle, and typically raise them asynchronously in application components. MagnetOS provides the necessary abstractions to specify applications composed of such events and event handlers, the static service required to partition such an application into its separate components, and a runtime that modifies the behavior of the application dynamically to optimize system goals such as energy usage.

MagnetOS applications are specified as regular Java programs which define component boundaries as well as provide well-typed event specifications. This enables the

bulk of the application logic to be expressed using familiar Java syntax and semantics. The MagnetOS rewriting engine partitions monolithic applications and distributes their components across an ad hoc network. The MagnetOS runtime then coordinates the communication and migration of these application segments across the nodes in the ad hoc network in order for the newly distributed application to execute in a power-efficient manner. We discuss the static and dynamic components of the MagnetOS runtime in the following sections.

3.1 Static Partitioning

MagnetOS partitions applications based on programmer annotations, though, in the absence of annotations, object boundaries delineate event handlers. Consequently, the unit of mobility in MagnetOS is typically a Java object instance, which we use synonymously with event handler. This transformation at class boundaries preserves existing object interfaces, and inter-object invocations define events in MagnetOS. The entire transformation is performed at the byte-code level via binary rewriting, without requiring source-code access.

Our approach to partitioning applications statically is patterned after previous work on application rewriting at ingress points [42]. Static partitioning confers several advantages. First, the complex partitioning services need only be supported at code-injection points, and can be performed offline. Second, since the run-time operation of the system and its integrity do not depend on the partitioning technique, users can partition their applications into arbitrary components if they so choose. Further, since applications are verified prior to injection into the network, individual MagnetOS nodes need not rerun a costly verifier on application components.

3.1.1 Application Partitioning for the JVM

Static partitioning takes original application classes, and from each class extracts an event handler, a dispatch handle, an event descriptor, and a set of event globals associated with the event handler.

An event handler is a modified implementation of the original class that stores the instance variables of the corresponding object. Each handler is free to move across nodes in the network. Dispatch handles, on the other hand, are remote references through which components can raise events. That is, dispatch handles are used to invoke procedure calls on remote event handlers residing on other nodes. Event raises through the dispatch handle are intercepted by the MagnetOS runtime and converted into RPCs. This indirection enables code migration. As an event handler moves, the event raises occurring through the corresponding event dispatch handles are tracked by the MagnetOS runtime and directed to the new location of the event handler. Event descriptors capture the event signatures that the original code exposes to the rest of the application.

Several modifications to the application binaries are required for this remote object mechanism to work seamlessly. First, object creations (new instructions and matching constructor invocations) are replaced by calls to the local MagnetOS runtime. The runtime selects an appropriate node and constructs a new event handler at that location. This operation returns a corresponding, properly initialized dispatch handle, which is then used in subsequent event raises. In addition, MagnetOS converts remote data accesses into events corresponding to accessor functions to read and write named locations. Similarly, it converts lock acquisitions and releases into centralized operations at the event handler. Finally, typechecking and synchronization instructions (check-cast, instanceof, monitorenter and monitorexit instructions, and synchronized methods) are rewritten to trap into the MagnetOS runtime.

The final component created for a class is a set of event globals. The event globals are static fields shared across all instances of an event handler. Each event handler retains pointers to the corresponding instance of event globals, and can therefore share state with other handlers.

3.2 Dynamic Object Management

The MagnetOS runtime provides the dynamic services that facilitate the distributed execution of componentized applications across an ad hoc network. Its services include component creation, inter-component communication, migration, garbage collection, naming, and event binding.

3.2.1 Object Creation

In order to create a new instance of an event handler, an application contacts the local runtime and passes the requisite type descriptor and parameters for creation. The runtime then has the option of placing the newly created handler at a suitable location with little cost. It may choose to locate the handler on the local node, at a well-known node or at its best guess of an optimal location within the network. In our current implementation, all new handlers are created locally. We chose this approach for its simplicity, and rely on our dynamic migration algorithms to find the optimal placement over time. Furthermore, short-lived, tightly scoped event handlers do not travel across the network unnecessarily. The application binaries, containing all of the constructors, are distributed to all nodes at the time that the application is introduced into the network. Once created, the (remote) runtime simply initializes the handler by calling its constructor and returns a dispatch handle.

Every object is tagged with a unique id, consisting of a (node id, object id) tuple, as well as a creation time derived from a Lamport clock kept independently at each node. The unique id can be generated without expensive distributed consensus, and the creation time enables ob-

ject versions to be differentiated from each other, aiding in failure detection and recovery.

3.2.2 Remote Invocation and Migration

The runtime transparently handles invocations among the event handlers distributed across the network. Each runtime keeps a list of local event handlers. Dispatch handles maintain the current location of the corresponding handler, and processes raise events on behalf of application invocations by marshalling and unmarshalling event arguments and results.

MagnetOS can migrate both active and passive event handlers. A passive event handler consists purely of static data associated with an event handler; since no active computation is modifying the handler's associated data, it is migrated at run time by serializing handler state and moving it to a new node. In order to reduce the energy cost of migration, MagnetOS informs dispatch handles of relocation lazily, the next time they raise an event or the next time the garbage collector renews their object leases on remote objects. This notification is accomplished through forwarding references left behind when event handlers migrate. Chains of forwarding pointers, if allowed to persist for a long time, would pose a vulnerability - as nodes die, out-of-date event references may lose the path to the current location of the handler to which they are bound. MagnetOS collapses these paths whenever they are traversed. Periodic lease updates in lease-based garbage collection requires periodic communication between dispatch handles and event handlers, which provides an upper-bound on the amount of time such linear chains are permitted to form in the network.

Migrating active event handlers efficiently requires effectively capturing the current state of the computation being performed by that handler, and relocating it to a new node. Since this state may be large, migration of active event handlers is a costly operation. MagnetOS takes into account the cost of the migration when making a decision to relocate a handler. In some cases, such as a node whose battery level is below a critical threshold required to offload all event handlers at that node, active migration is invoked in order to preserve the computation without disruption. In order to provide the migration of active handlers without incurring excessive run time costs, the MagnetOS static partitioning service injects code to periodically check a flag in each basic block. If the handler has been identified for migration, the MagnetOS runtime sets this flag. When the rewritten code detects that the flag is set, it checkpoints the current state of computation and traps into MagnetOS runtime. MagnetOS runtime transports the state and resumes the computation at the destination. Compared to other approaches based on periodic checkpointing like JavaGoX[40] or state tracking on a parallel stack like Merpati[45], this approach, similar to Brakes[48], incurs the cost of object checkpointing only when the event handler needs to be migrated.

3.2.3 Handling Failures

Executing an application in a distributed setting introduces a new set of failure modes. Since attempting to preserve single-system application semantics in the presence of such failures is futile due to well-known impossibility results, MagnetOS provides reasonable defaults for common cases and exposes the unrecoverable errors to applications. The system provides traditional at-most-once semantics for event invocation; namely, events are guaranteed to execute at most once, and failure indications raised by the run time kernel are conservative.

The main insight guiding failure detection in the ad hoc domain with a single system image is that failure detection can be deferred until a node requires the results of a computation executed elsewhere. Consequently, frequent pings to check on the availability of other nodes, common in the wide area setting though costly from an energy perspective in an ad hoc network, can be avoided. MagnetOS performs health checks only when a node is waiting on the result of a computation from another node. The system uses keep-alive messages for long-running synchronous event invocations; the loss of a threshold number of consecutive keepalives indicate a failure to communicate between the two nodes, stemming from the MAC, transport, routing, or MagnetOS migration layers.

In response to such failures, MagnetOS attempts to mask as much as it can without violating application semantics. MAC layer failures, due to interference, congestion, or lack of connectivity, are retried at the transport layer. The routing layer is closely coupled to the transport layer and initiates a route discovery when triggered by the transport layer. The migration layer uses forwarding pointers to track objects that have moved, and when the pointer chain is broken due to a failure, falls back to a broadcast query for the target event handler as a last resort. Despite these measures, nodes that are disconnected may well experience unrecoverable errors.

Unrecoverable errors are reflected to applications as a special run time exception, one that can be implicitly raised by any procedure in the system, that notifies a caller that an event invocation has failed. The caller can catch such exceptions and take remedial action, taking care that the failure detectors are conservative; that is, an event may have been invoked, and the failure may well have occurred in notifying the invoker of the successful completion of the event. Each event invocation carries a unique identifier that can be used readily to guard against multiple invocations of a handler. In some cases, a network failure may have resulted in a portion of the application being broken off the rest of the network, along with corresponding application state. When the broadcast query for an event handler fails, the application is notified that such a failure has occurred. In response,

applications that can tolerate some loss of state can regenerate new objects, while others can fail with a hard failure. The unique identifier and creation time attached to every object by its creating node allows applications to detect version mismatches. The migration algorithms described later keep track of the identifiers and version numbers of objects that contacted a given object within an epoch, thereby allowing the system to cheaply detect a mismatch subsequent to a network partition and recovery. Alternatively, applications can perform their own global disconnect recovery algorithms using the same object version information. In all cases, an exception is thrown and application specific recovery code supplied by the programmer can be invoked. Determining application semantics in order to recover automatically from complicated failures is a difficult problem; MagnetOS strikes a balance by asking for programmer direction in case of object loss and healed network partitions.

3.2.4 Explicit Overrides

The MagnetOS runtime provides an explicit interface by which application writers can manually direct component placement. This interface allows programmers to establish affinities between event handlers and ad hoc nodes. We provide two levels of affinity. Specifying a *strong* affinity between an event handler and a node effectively anchors the code to that node. This is intended for attaching event handlers like device drivers to the nodes with the installed device in them. Specifying a *weak* affinity immediately migrates the component to the named node, and allows the automated code placement techniques described in the next section to adapt to the application's communication pattern from the new starting point. Note that today's manually constructed applications correspond to the use of strong affinity in our system - unless explicitly moved, components are bound to nodes. The result of overusing strong affinity is a fragile system, where unforeseen communication and mobility patterns can leave an application stranded. While we provide these primitives in order to ensure that MagnetOS applications provide at least as much control to the programmer as manually crafted applications, we do not advocate their use.

3.3 Event Handler Placement

In this section, we describe the algorithms we developed to automatically find an energy-efficient placement for application components.

All of our algorithms share the same basic insight: they shorten the mean path length of data packets by moving communicating objects closer together. They do this by profiling the communication pattern of each application in discrete time units, called epochs. In each epoch, every runtime keeps track of the number of incoming and outgoing packets for every object. At the end of each epoch, the migration algorithm decides whether to move that object, based on its recent behavior. The placement decision is made locally, based on information collected during recent epochs at that node.

LinkPull collects information about the communication pattern of the application at the physical link level, and migrates components over physical links one hop at a time. This requires very little support from the network; namely, the runtime needs to be able to examine the link level packet headers to determine the last or next hop for incoming and outgoing packets, respectively. For every object, we keep a count of the messages sent to and from each neighboring node. At the end of an epoch, the runtime examines all of these links and the object is moved one hop along the link with greatest communication.

PeerPull operates at the network level, and migrates components multiple hops at a time. In each epoch, PeerPull examines the network source addresses of all incoming messages, and the destination addresses of outgoing messages for each object. This information is part of the transmitted packet, and requires no additional burden on the network. At the end of an epoch, PeerPull finds the host with which a given object communicates the most and migrates the object directly to that host.

NetCluster migrates components to a node in the most active cluster. Nodes that share the same next or last hop on the route are considered to be in the same cluster. At the end of each epoch, NetCluster finds the cluster that a object communicates with the most and then migrate the object to a randomly chosen node within that cluster.

TopoCenter(1) migrates components according to a partial view of the network built at each node, based on connectivity information attached opportunistically to each packet. Each node attaches its single-hop neighborhood to outgoing packets, subject to space constraints. The topology information is accumulated along the path to the destination. Nodes on the path extract the topology information from packets they forward. At the end of an epoch, each node computes the shortest paths between itself and other nodes in its partial network view and moves an object to the node such that the sum of migration cost (proportional to the shortest path) and estimated future communication cost (based on the packet statistics of the last epoch) is minimized. TopoCenter(Multi) is similar to TopoCenter(1), except that nodes attach all the topology information they know to outgoing packets.

All of these algorithms improve system longevity by using the available power within the network more effectively. By migrating communicating components closer to each other, they reduce the total distance packets travel, and thereby reduce the overall power consumption. Further, moving application components from node to node helps avoid hot spots and balances out the communication load in the network. As a result,

these algorithms can significantly reduce power consumption and improve the total system longevity for energy-constrained ad hoc networks.

3.4 MagnetOS API

MagnetOS provides several key abstractions for programmers to be able to implement distributed ad hoc networking applications effectively. These abstractions enable applications to name nodes and application components, to collect statistics and information on network and node behavior, and to set and direct migration policies. In addition, some existing Java abstractions are amended to reflect their new functionality and failure modes in the distributed MagnetOS setting. We describe the significant parts of the API below.

A **Node** encapsulates the notion of a physical host that is running a MagnetOS instance. It enables applications to name and query nodes. A node handle can be used to migrate application components, to anchor an object to a specific location in the network, and to query node properties such as link state, network topology information and energy status.

The **Link** abstraction refers to a physical link between two Nodes and the **NeighborSet** abstraction is the set of single hop neighbors of a given Node. These two abstractions enable applications to discover the network topology and link characteristics based on low-level information that is being updated by the operating system..

Energy enables applications to query the energy level of the underlying platform. Applications typically take actions in face of changing of battery energy. The Energy API allows applications to query the current energy level, the drain rate, and recharge frequency of the underlying heterogeneous platforms through a common interface.

A **Timer** abstraction enables applications to schedule events to be invoked at a predetermined time in the future. Timers can be migrated between nodes unless specifically pinned by the application on a specified node. A closure, saved along with the timer at the time of registration, provides the necessary context for event arguments when the timer fires.

Lock is a remote synchronization object, analogous to the monitor in Java. Standard monitor and condition variables can only be used locally. In MagnetOS, Lock acquisitions and releases are converted into remote operations on the event handler. Identifiers used for threads are replaced by globally unique identifiers for active event handlers, and keep-alives are added between the remote lock holder and the lock instance in order to detect network failures. In case of such a failure, a RemoteLockable object is restored to its state saved at the point of lock acquisition and the lock is released with an exception. In order to guard this scheme from leading to inconsistent application state in the presence of nested locks (i.e. where a failure rolls back the state of one, but not all objects), MagnetOS enables applications to override the acquisition and releasing methods. For instance, if a RemoteLockable object is stateless, the Lock acquisition method can be overridden such that no checkpointing is made, and if the recovery requires rollback of multiple objects, the restore method can perform the necessary cleanup.

A **Thread** in MagnetOS is an execution context that can perform a sequence of synchronous event raises. It differs from threads in standard Java in that it is distributed over multiple nodes and can migrate from one node to another while Java threads are confined to a single node. Each Thread in MagnetOS has a unique identifier, which is crucial for the remote synchronization mechanism in MagnetOS.

All of these abstractions simplify the application development, gives programmers more flexibility, and enables applications to take advantage of information already kept by lower software layers in the MagnetOS runtime.

3.5 Implementation Support for Ad hoc Networks

The ad hoc networking domain places additional constraints on the runtime implementation. First, multihop ad hoc networks require an ad hoc routing protocol to connect non-neighboring nodes. MagnetOS relies on a standard ad hoc routing protocol below the runtime to provide message routing. Currently, our system runs on any platform that supports Java JDK1.4. On Linux, we use an efficient in-kernel AODV implementation we developed. On other platforms, we use a user-level version of AODV written in Java to provide unicast routing. The choice of a routing algorithm is independent from the rest of the runtime, as the runtime makes no assumptions of the routing layer besides unicast routing.

Second, standard communication packages such as Sun's RMI are designed for infrastructure networks, and are inadequate when operating on multihop ad hoc networks. Frequent changes in network topology and variance in available bandwidth require MagnetOS to migrate objects, requiring the endpoints of an active connection to be modified dynamically as objects move. We have built a custom RPC package based on a reliable datagram protocol [21] that allows us to easily modify the communication endpoints when components move and is responsible for all communication between dispatch handles and corresponding event handlers.

Finally, the higher-level policies in MagnetOS require information on component behavior to make intelligent migration decisions. The runtime assists in this task by collecting, for each component, information on the amount of data it exchanges with other components. The runtime intercepts all communication and records the source and destination for all incoming and outgoing events. MagnetOS keeps a cumulative sum per compo-

nent per epoch, and periodically informs the migration policy in the system of the current tally. While this approach has worst case space requirement that is $O(N^2)$, where N is the number of components in the network, in practice most components communicate with few others and the space requirements are typically small. For instance, in the sensor benchmark examined in Section 4, the storage requirements are linear. The next section describes how MagnetOS uses these statistics to automatically migrate components.

4 Evaluation

In this section, we examine the power efficiency of automatic migration strategies in MagnetOS. We first evaluate the core automatic migration algorithms, LinkPull, PeerPull, NetCluster, TopoCenter(1), and TopoCenter(Multi), in three different benchmarks. We compare our algorithms with manual (Static algorithm), natural load balancing (Random algorithm), and optimal (See definition below), and show that they achieve good energy utilization, improve system longevity, and are thus suitable for use in general-purpose, automatic migration systems. Next, we report results from microbenchmarks to show that automatically partitioning applications does not extract a large performance cost. We also present evidence showing that the memory costs of a specially tuned Java virtual machine is within the resource-budget of next generation ad hoc nodes. Finally, we show through a physical deployment that the simulated results match those observed in the real world.

4.1 Benchmarks and Workload

We evaluated the performance and efficiency of MagnetOS event handler migration strategies in three representative applications, each with a unique communication pattern and application workload. The applications were chosen to span a wide range of possible deployment environments. We first describe the setup and workload for each application, then examine their performance under MagnetOS.

4.1.1 SenseNet

We first examine a generic, reconfigurable sensing benchmark we developed named SenseNet. This application consists of sensors, condensers and displays. Sensors are fixed at particular ad hoc nodes, where they monitor events within their sensing radius and send a packet to a condenser in response to an event. Condensers can reside on any node, where they process and aggregate sensor events and filter noise. The display runs on a well-equipped central node, extracts high-level data on events from the condensers, and sends results to an external wired network.

The application is run on a 14 by 14 grid of sensors, each placed 140 meters apart with a uniformly distributed jitter of 50 meters. The communication and sensing radius is 250 meters. The grid is partitioned into four quadrants, and a single condenser is assigned to aggregate and process data for each quadrant. The workload consists of two bodies that move through the sensor grid in randomly chosen directions. We measure the total remaining energy across all nodes, sensor coverage, number of drained sensor nodes, number of nodes not reachable by the display, and overall system longevity. We define system failure as the point when half of the field is no longer being sensed by sensors connected to the display node.

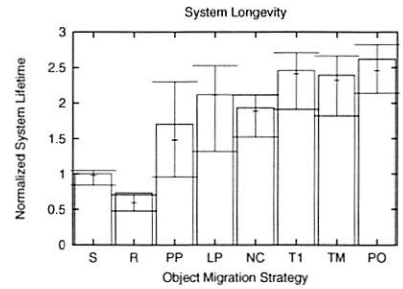


Figure 3: Automatic migration significantly extends system lifetime in the SenseNet application. Bars represent 25th and 75th quartiles.

4.1.2 Publish-Subscribe

Our second application consists of a basic publish-subscribe system. The application provides a channel abstraction to which clients can subscribe and publish. Channels act as mobile rendezvous points by accepting incoming messages and relaying them to each of the clients subscribed to the channel.

For this application, we generate a workload resembling a disaster recovery application. The workload consists of ten channels each with four subscribers. The four subscribers publish messages approximately every 10, 20, 30, and 40 seconds, respectively. We run the application on the same network layout as SenseNet. We measure total system throughput smoothed over 20 second intervals, number of nodes drained, and total remaining energy in the network. We stop the simulations when total throughput drops to zero during a 20 second interval.

4.1.3 FileSystem

Our final benchmark is a network file system that may be used in mobile ad hoc scenarios. This application consists of clients and files. Client objects are assigned to mobile devices, and access files over the network according to an external trace. File objects can reside on any node, and independently receive and process requests from clients.

This application is run on a randomly generated network with 196 mobile nodes, with approximately the same density as in SenseNet. The nodes move according to the random waypoint mobility model with a maxi-

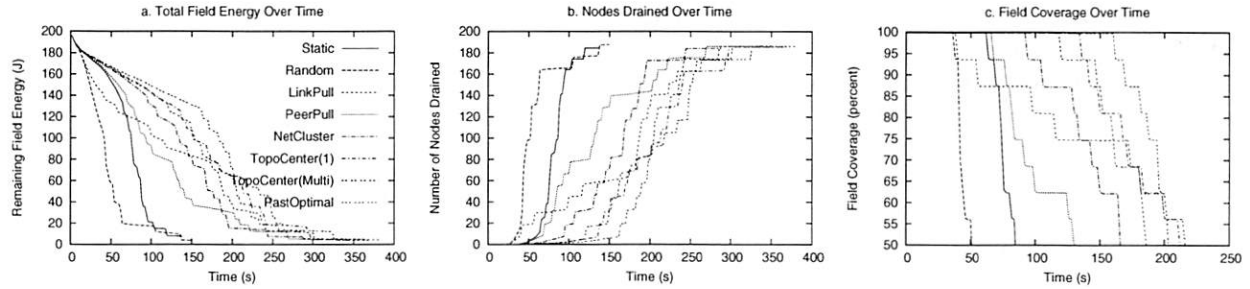


Figure 4: **SenseNet application:** Adaptive algorithms extract more energy out of the field and so increase field coverage and node availability.

mum node speed of 5 meters per second. The benchmark workload is based on the Auspex file system trace [10]. To compensate for the relatively limited capabilities of wireless nodes, we slow the trace by a factor of four. We measure the same statistics, and use the same stopping condition, as for the Publish-Subscribe application.

4.2 Simulation Methodology

We implemented a significant part of the MagnetOS system in **sns** [49], a scalable version of the Ns-2 network simulator. In order to accurately account for packet-level costs, we implemented a detailed energy model using parameters obtained from measurements of 802.11b wireless cards [12]. Computation costs are assumed to be negligible. We use the AODV protocol for wireless ad hoc routing, which includes support for both mobile and static node placements, and include the cost for route discovery, maintenance, and repair in our energy model. In all, we run each application with 16 scenarios each by varying the workload and network layout, averaging the results to obtain estimates of expected application behavior. For each application and scenario, we compare our algorithms described in section 3.3 with the following object placement and migration strategies:

Static Centralized corresponds to a static, fixed assignment of all movable objects to a single, central node in the network. All movable components remain at the home node for the entire duration of the simulation. **Random** selects a random neighbor as destination for each movable object at each epoch. It corresponds to a simple load-balancing algorithm, designed to avoid network hotspots. **PastOptimal** assumes that every node in the network has full knowledge of the network topology and moves objects to an optimal position based on the communication pattern in the last epoch. PastOptimal does not necessarily form the upper bound for other algorithms because it is still based on past behavior, and it may make bad decisions if the communication pattern changes significantly in the next epoch. Nevertheless, it is a good benchmark for migration algorithms based on past behavior of an object.

In addition to simulation-based evaluation, we implemented these benchmarks on top of our prototype system that supports x86/Windows, x86/Linux, and StrongArm/PocketPC platforms. The base system includes adaptive object placement policies, AODV ad hoc wireless routing, and automatic partitioning using Java bytecode rewriting. We will validate the simulation results against our physical test result later in this section.

4.3 Results and Discussion

The benchmarks represent a wide spectrum of different applications and communication models, and thus the relative performance of static and intelligent object migration policies varies with the application. Overall, these benchmarks show that the adaptive algorithms described above avoid hotspots in the network by moving objects intelligently. In addition, we find that the details of application communication and workload patterns impact the relative performance of different migration strategies, confirming the need for automatic and system-wide placement policies.

4.3.1 SenseNet

The SenseNet benchmark shows the clearest gains for the adaptive algorithms described above. Figure 3 shows that automatic migration increases system longevity by a factor of 2.5. This gain is achieved generally by moving objects away from hotspots and reducing mean packet distances. TopoCenter(Multi) performs a little worse than TopoCenter(1), which suggests that for this specific application, one-hop topology is good enough to make intelligent migration decisions and carrying multi-hop topologies will only increase the packet size and overhead, and decrease the system longevity.

Figure 4 shows SenseNet behavior in greater detail. The energy curves in (a) for the adaptive algorithms are more shallow than those for the Random and Static cases, which are steep and linear in the time of the simulation. Static suffers because of the energy bottleneck it creates around the fixed locations it has of system components. Random, a standard approach to distribute load, actually hurts energy performance by paying too much in migration costs.

The unreachable nodes (b) and coverage (c) graphs show two separate performance metrics with similar insights. Adaptive placement and migration save energy

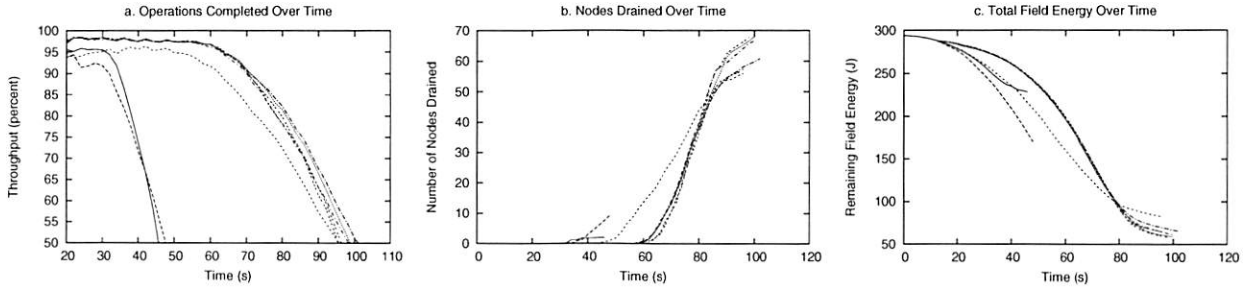


Figure 5: **Publish-Subscribe benchmark.** Adaptive algorithms improve throughput, availability and energy utilization.

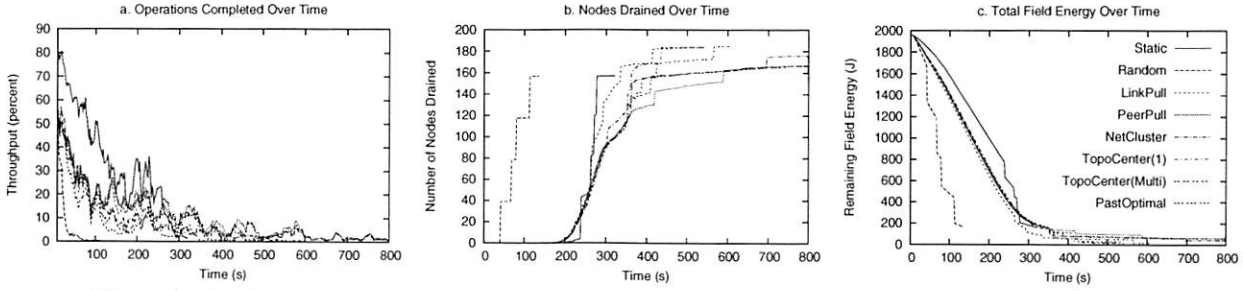


Figure 6: **FileSystem benchmark.** Comparison of adaptive algorithms and static and random load balancing.

and distribute load, which extends node lifetimes and increases longevity for the network.

4.3.2 Publish-Subscribe

The Publish-Subscribe application differs substantially from the SenseNet application. It consists of a relatively small number of rendezvous points, each communicating with a stable set of clients. This enables route discovery costs, incurred when objects are migrated, to be amortized over a large number of accesses.

Figure 5a shows that the Centralized approach fails very early because of the large hotspot in client accesses around the center of the network. We can see in each of the graphs that this approach initially achieves as high throughput as the dynamic migration strategies, but it dies early when it uses up the energy in the center of the network. The Random approach shows that randomizing the location of objects fails to achieve any savings, as it incurs the migration cost without the benefits of intelligently placing objects in the network.

MagnetOS policies do significantly better, since they place the rendezvous points near clients that access them frequently, reducing the ongoing cost of publish-subscribe operations. LinkPull does not do as well as the other adaptive algorithms because it requires several epochs to move the rendezvous points to the clients when subscriptions change. All other algorithms are very close to PastOptimal in their power performance.

4.3.3 FileSystem

In our file system benchmark, the centralized static algorithm performs very well initially, but fails afterwards with a steep cliff. The main reason for this is that Static avoids the energy cost for migration and route discovery but fails to avoid the hotspot around the central file server. Node mobility helps mitigate the hotspots somewhat and helps Static perform better in this benchmark than in the other two, but Static nevertheless suffers eventually. The lower throughput of adaptive algorithms initially is mainly due to changes of file locations. File operations following file migration require new route discovery, which increases latency and reduces the throughput.

The benefit of using an adaptive algorithm is clearly seen in the number of nodes drained (Figure 6b), where adaptive algorithms manages to spread the drain more evenly over the nodes of the network. The simple PeerPull algorithm performs surprisingly close to the PastOptimal algorithm. The Static and Random algorithms cannot do so intelligently and incur large losses of nodes due to hotspots.

Figure 6c shows similar results. The Static algorithm does well initially because it does not incur any costs for migration and route discovery. The centralized algorithm only has a single destination for all data flows. This layout is an optimal case for the AODV routing layer. All the adaptive cases, by event handler migration, have individual files which are located at many different points in the network. This leads to hundreds of different destinations, a load which is significantly more expensive for AODV to compute and maintain. Although adaptive algorithms consume more energy overall, that energy consumption is spread out over the whole network.

This benchmark shows that in applications where workload is comparatively low and routing cost dominates the overall energy consumption, adaptive algorithms can still help by distributing the energy cost throughout the network and avoiding the hotspots around central nodes.

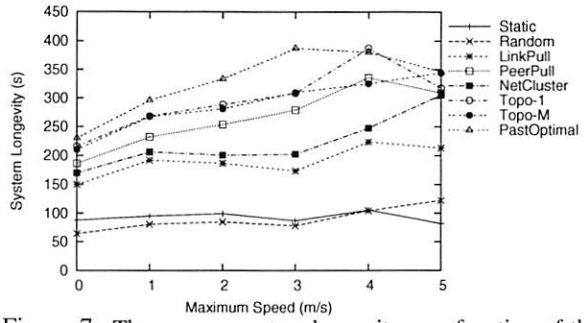


Figure 7: The average system longevity as a function of the maximum node speed.

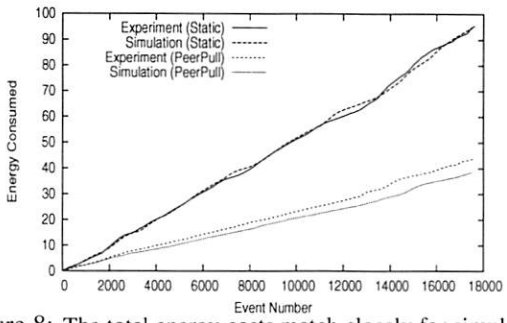


Figure 8: The total energy costs match closely for simulation and physical test results

Remote Call	Java RMI	MagnetOS
Null	403±16	172±6
Int	446±9	180±8
Obj. with 32ints	991±35	174±4
Obj. with 4int,2obj	884±21	177±7

Table 1: Remote method invocation comparison. All times in μs, average of 1000 calls.

4.3.4 Node Mobility

We evaluate the effect of node mobility on the performance of MagnetOS using the SenseNet benchmark. The simulation setup is modified to incorporate node movement according to the random waypoint mobility model with a pause time equal to $(50/S)$ seconds, where S is the maximum node speed. We vary the value of S and measure the average system longevity for each algorithm. Figure 7 shows that system longevity increases initially with mobility as physical movement acts as a simple load balancing scheme that avoids hotspots. Mobility also increases the route rediscovery cost and decreases the effectiveness of the prediction of our adaptive algorithms. However, adaptive algorithms perform well overall in mobile scenarios.

4.4 Validation

We validate the simulation results against our system implementation. We use the SenseNet benchmark for validation. Our implementation code executes on Linux 2.4.18 on Dell inspiron 2650 laptops, each equipped with Celeron M 1.5GHz processor, 256MB memory and Dell TrueMobile 1150 series 11Mbps Wireless LAN Adapters. To exclude the energy consumption of other devices like CPU, memory and LCD display, we change the wireless card driver to calculate energy cost of each packet transmission and reception.

The experiment setup is similar to the simulation setup we discussed above except we decrease the scale down to 16 nodes. 16 laptops are set up in a 4 by 4 grid, each running the MagnetOS runtime. The grid is partitioned into four quadrants, and a single condenser is assigned to aggregate and process data for each quadrant. Two objects move through the sensor grid in randomly chosen directions and speeds, triggering audio sensors on Magnetos nodes. In order to match the time scales of the physical experiment to the simulation, we measure the total energy cost of the field per event sensed by the sensor. We compare the results of the Static and PeerPull migration policies. Figure 8 shows that the energy consumption of the physical experiment closely match those of the simulations. The differences are attributable to variations in physical packet sizes from the average object size assumed in simulations.

4.5 Space and Time Overhead

An automatic approach to application partitioning and transparent object migration would be untenable if the performance of automatically partitioned applications suffered significantly. In the micro-benchmark shown in Table 1, we compare the overhead of our RPC implementation to that of remote invocations performed via Java RMI, on a 1.7 GHz P4 with 256 MB of RAM JDK 1.4 on Linux 2.4.17 with AODV. On all micro-benchmarks, automatically decomposed applications are competitive with manually coded, equivalent RMI implementations, due partly to tight integration of system code with application code through binary rewriting.

Traditional Java VMs, such as Sun's JDK, support many features and are not optimized for space. Consequently, they require large amounts of memory and are not suitable for resource constrained nodes. For MagnetOS, we have developed our own JVM. and PocketPC/StrongArm devices. This JVM works on x86 laptops and PocketPC/StrongARM devices and performs space optimizations that include lazy class loading, constant pool sharing, stack and memory compression, and aggressive class unloading. Our JVM reduces the memory required to run MagnetOS from over 9 MB to approximately 1350 KB. This is well within the memory budget of existing mobile devices, and within a few years of Moore's Law growth of sensor nodes.

4.6 Summary

In this section, we examined three benchmarks built under the MagnetOS single system image model, and evaluated the performance of automatic migration strate-

Low overhead solutions using differential patching are proposed in [20] and [11]. In these schemes a binary differential update between the original system image and the new image is created. A simple language is used to encode this update in a compressed form that is then distributed through the network and used to construct a new system image on deployed nodes. This technique can be used at the component level in SOS and, since changes to the SOS kernel or modules will be more localized than in tightly coupled systems, result in smaller differentials.

MOAP [22] and Deluge [10] are two protocols that hove been used to distribute new system images to all nodes in a sensor network. SOS currently includes a publish-subscribe scheme that is similar to MOAP for distributing modules within a deployed sensor network. Depending on the user's needs SOS can use MOAP, Deluge, or any other code distribution protocol. SOS is not limited to running the same set of modules or same base kernel on all nodes in a network, and will benefit from different types of distribution protocols including those supporting point-to-point or point-to-region transfer of data.

3 System Architecture

In addition to the traditional techniques used in embedded system design, the SOS kernel features dynamically linked modules, flexible priority scheduling, and a simple dynamic memory subsystem. These kernel services help support changes after deployment, and provide a higher level API freeing programmers from managing underlying services or reimplementing popular abstractions. Most sensor network application and protocol development occurs in modules that sit above the kernel. The minimal meaningful sensor network application using SOS consists of a simple sensing module and routing protocol module on top of the kernel.

Table 1 presents memory footprints of the core SOS kernel, TinyOS with Deluge, and Maté Bombilla virtual machine. All three of these configurations include a base operating environment with the ability to distribute and update the programs running on sensor nodes. SOS natively supports simple module distribution and the ability to add and remove modules at run time. TinyOS is able to distribute system images and reflash nodes using Deluge. The Maté Bombilla VM natively supports the transfer and execution of new programs using the Trickle [16] protocol. SOS is able to provide common kernel services to external modules in a footprint comparable to TinyOS running Deluge and a space smaller than the Maté Bombilla VM. Note that RAM usage in SOS is broken into two parts: RAM used by the core kernel and RAM reserved for the dynamic memory subsystem.

The following discussion takes a closer look at the key architectural decisions in SOS and examines them in light of the commonly used TinyOS.

Platform	ROM	RAM
SOS Core	20464 B	1163 B
(Dynamic Memory Pool)	-	1536 B
TinyOS with Deluge	21132 B	597 B
Bombilla Virtual Machine	39746 B	3196 B

Table 1—Memory footprint for base operating system with ability to distribute and update node programs compiled for the Mica2 Motes.

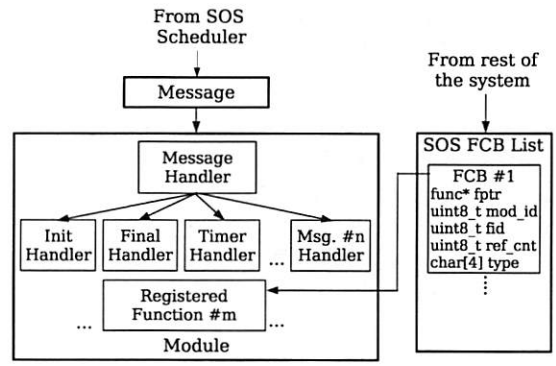


Figure 1—Module Interactions

3.1 Modules

Modules are position independent binaries that implement a specific task or function, comparable in functionality to TinyOS components. Most development occurs at the module layer, including development of drivers, protocols, and application components. Modification to the SOS kernel is only required when low layer hardware or resource management capabilities must be changed. An application in SOS is composed of one or more interacting modules. Self contained position independent modules use clean messaging and function interfaces to maintain modularity through development and into deployment. The primary challenge in developing SOS was maintaining modularity and safety without incurring high overhead due to the loose coupling of modules. An additional challenge that emerged during development, described in more detail in section 3.1.4, was maintaining consistency in a dynamically changing system.

3.1.1 Module Structure

SOS maintains a modular structure after distribution by implementing modules with well defined and generalized points of entry and exit. Flow of execution enters a module from one of two entry mechanisms: messages delivered from the scheduler and calls to functions registered by the module for external use. This is illustrated in figure 1.

Message handling in modules is implemented using a module specific handler function. The handler function

Communication Method	Clock Cycles
Post Message Referencing Internal Data	271
Post Message Referencing External Buffer	252
Dispatch Message from Scheduler	310
Call to Function Registered by a Module	21
Call Using System Jump Table	12
Direct Function Call	4

Table 2—Cycles needed for different types of communication to and from modules in SOS when running on an Atmel AVR microcontroller. The delay for direct function calls within the kernel is listed as a baseline reference.

takes as parameters the message being delivered and the state of the module. All module message handlers should implement handler functions for the *init* and *final* messages produced by the kernel during module insertion and removal, respectively. The *init* message handler sets the module's initial state including initial periodic timers, function registration, and function subscription. The *final* message handler releases all node resources including timers, memory, and registered functions. Module message handlers also process module specific messages including handling of timer triggers, sensor readings, and incoming data messages from other modules or nodes. Messages in SOS are asynchronous and behave somewhat like TinyOS tasks; the main SOS scheduling loop takes a message from a priority queue and delivers the message to the message handler of the destination module. Inter-module direct function calls are used for module specific operations that need to run synchronously. These direct function calls are made possible through a function registration and subscription scheme described in section 3.1.2. Module state is stored in a block of RAM external to the module. Modules are relocatable in memory since: program state is managed by the SOS kernel, the location of inter-module functions is exposed through a registration process, and the message handler function for any module is always located an a consistent offset in the binary.

3.1.2 Module Interaction

Interactions with modules occur via messages, direct calls to functions registered by a module, and ker_* system calls into the SOS kernel. The overhead for each of these types of interactions is presented in table 2. Messaging, detailed in section 3.2, provides asynchronous communication to a module and enables scheduling by breaking up chains of execution into scheduled subparts. Messaging is flexible, but slow, so SOS provides direct calls to functions registered by modules, which bypass the scheduler to provide low latency communication to modules.

Function registration and subscription is the mechanism that SOS uses to provide direct inter-module communication and upcalls from the kernel to modules.

Function Registration Action	Clock Cycles
Register a Function	267
Deregister a Function	230
Get a Function Handle	124
Call to Function Registered by a Module	21

Table 4—Cycles needed for the function registration mechanism in SOS when running on an Atmel AVR microcontroller.

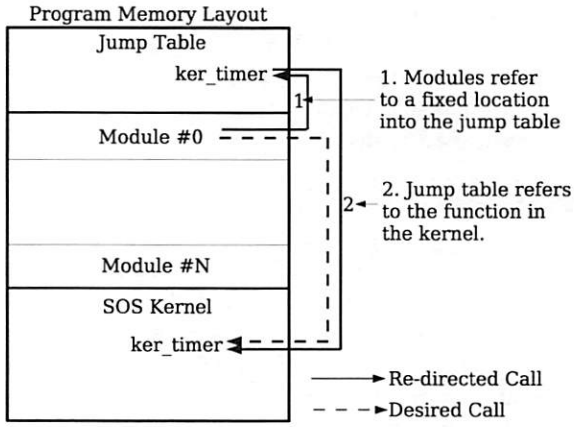


Figure 2—Jump Table Layout and Linking in SOS.

When explicitly registering functions with the SOS kernel, a module informs the kernel where in its binary image the function is implemented. The registration is done through a system call *ker_register_fn* described in table 3 with overheads detailed in table 4. A function control block (FCB) used to store key information about the registered function is created by the SOS kernel and indexed by the tuple {module ID, function ID}.

The FCB includes a valid flag, a subscriber reference count, and prototype information. The stored prototype information encodes both basic type information and whether a parameter contains dynamic memory that needs to undergo a change of ownership. For example, the prototype {'c', 'x', 'v', '1'} indicates a function that returns a signed character ('c') and requires one parameter ('1'). That parameter is a pointer to dynamically allocated memory ('x'). When a registered function is removed, this prototype information is used by kernel stub functions described in section 3.1.4.

The call *ker_get_handle* described in table 3 is used to subscribe to a function. The module ID and function ID are used as a tuple to locate the FCB of interest, and type information is checked to provide an additional level of safety. If the lookup succeeds, the kernel returns a pointer to the function pointer of the subscribed function. The subscriber should always access the subscribed function by dereferencing this pointer. This extra level of indirection allows the SOS kernel to easily replace the implementation of a function with a newer version by changing the function pointer in the FCB, without needing to update subscriber modules.

Prototype	Description
int8_t ker_register_fn(sos_pid_t pid, uint8_t fid, char *prototype, fn_ptr_t func)	Register function 'func' with type 'prototype' as being supplied by 'pid' and having function ID 'fid'.
fn_ptr_t* ker_get_handle(sos_pid_t req_pid, uint8_t req_fid, char *prototype)	Subscribe to the function 'fid' provided by module 'pid' that has type 'prototype'.

Table 3—Function Registration and Subscription API

Modules access kernel functions using a jump table. This helps modules remain loosely coupled to the kernel, rather than dependent on specific SOS kernel versions. Figure 2 shows how the jump table is setup in memory and accessed by a module. This technique also allows the kernel to be upgraded without requiring SOS modules to be recompiled, assuming the structure of the jump table remains unchanged, and allows the same module to run in a deployment of heterogeneous SOS kernels.

3.1.3 Module Insertion and Removal

Loading modules on running nodes is made possible by the module structure described above and a minimal amount of metadata carried in the the binary image of a module.

Module insertion is initiated by a distribution protocol listening for advertisements of new modules in the network. When the distribution protocol hears an advertisement for a module, it checks if the module is an updated version of a module already installed on the node, or if the node is interested in the module and has free program memory for the module. If either of the above two conditions are true, the distribution protocol begins to download the module and immediately examines the metadata in the header of the packet. The metadata contains the unique identity for the module, the size of the memory required to store the local state of the module, and version information used to differentiate a new module version. Module insertion is immediately aborted should the SOS kernel find that it is unable to allocate memory for the local state of the module.

A linker script is used to place the handler function for a module at a known offset in the binary during compilation, allowing easy linking during module insertion. During module insertion a kernel data structure indexed by the unique module ID included in the metadata is created and used to store the absolute address of the handler, a pointer to the dynamic memory holding the module state, and the identity of the module. Finally the SOS kernel invokes the handler of the module by scheduling an *init* message for the module.

The distribution protocol used to advertise and propagate module images through the network is independent of the SOS kernel. SOS currently uses a publish subscribe protocol similar to MOAP.

Module removal is initiated by the kernel dispatching a *final* message. This message provides a module a final opportunity to gracefully release any resources it is holding and inform dependent modules of its removal. After the *final* message the kernel performs garbage collection by releasing dynamically allocated memory, timers, sensor drivers, and other resources owned by the module. As described in section 3.1.4, FCBs are used to maintain system integrity after module removal.

3.1.4 Potential Failure Modes

The ability to add, modify, and remove modules from a running sensor node introduces a number of potential failure modes not seen in static systems. It is important to provide a system that is robust to these potential failures. While still an area of active work, SOS does provide some mechanisms to minimize the impact of potential failure modes that can result from dynamic system changes. These mechanisms set a global error variable to help inform modules when errors occur, allowing a module to handle the error as it sees fit.

Two potential modes of failure are attempting to deliver a scheduled message to a module that does not exist on the node, and delivering a message to a handler that is unable to handle the message. In the first case, SOS simply drops the message addressed to a nonexistent module and frees dynamically allocated memory that would normally undergo ownership transfer. The latter case is solved by the individual modules, which can choose custom policies for what to do with messages that they are unable to handle. Most modules simply drop these messages, return an error code, and instruct the kernel to collect dynamically allocated memory in the message that would have undergone a change of ownership.

More interesting failure modes emerge as a result of intermodule dependencies resulting from direct function calls between modules, including: no correct implementation of a function exists, an implementation of a function is removed, an implementation of a function changes, or multiple implementations of a single function exist.

A module's subscription request is successful if there exists a FCB that is tagged as valid and has the same module ID, function ID and prototype as that in the subscription. Otherwise the subscription request fails and the module is free to handle this dependency failure as it wishes. Actions currently taken by various SOS modules include aborting module insertion, scheduling a later attempt to subscribe to the function, and continuing to

execute with reduced functionality.

A subscription to a function can become invalid should the provider module be removed. When the supplier of a function is removed from the system, SOS checks if any other modules have registered to use the function. If the registration count is zero then the FCB is simply removed from the system. If the registration count is greater than zero, a control flag for the FCB is marked as being invalid to prevent new modules from subscribing and the implementation of the function is redirected to a system stub function. The system stub function performs expected memory deallocations as encoded in the function prototype and sets a global error variable that the subscriber can use to further understand the problem.

Problems can arise after a module has subscribed to a function if the implementation of the function changes due to updates to the provider module. Similar to when a module is removed, functions provided by a module are marked invalid during an update. When the updated module finishes installing, it registers new versions of the provided functions. SOS assumes that a function registration with the same supplier module ID, function ID, and prototype of an already existent FCB is an update to the function implementation, so the existent FCB is updated and the control flag is returned to being valid. This automatically redirects subscribers to the new implementation of the function. Changes to the prototype of a provided function are detected when the new implementation of the function is registered by the supplying module. This results in the old FCB remaining invalid with old subscribers redirected to a system stub and a new FCB with the same supplier module ID and function ID but different prototype information being created. New subscribers with the new prototype information are linked to the new FCB, while any attempts to subscribe to the function with the old prototype fail.

It is important to note that when SOS prevents an error in the situations described above, it typically returns an error to the calling module. The programmer of the module is responsible for catching these errors and handling them as they see fit. Providing less intrusive protection mechanisms is a fascinating and challenging problem that continues to be worked on as SOS matures.

3.2 Message Scheduling

SOS uses cooperative scheduling to share the processor between multiple lines of execution by queuing messages for later execution.

TinyOS uses a streamlined scheduling loop to pop function pointers off of a FIFO message queue. This creates a system with a very lean scheduling loop. SOS instead implements priority queues, which can provide responsive servicing of interrupts without operating in an interrupt context and more general support for passing parameters to components. To avoid tightly integrated modules that carefully manage shared buffers, a result of the inability to pass parameters through the messaging mechanism, messaging in SOS is designed to handle the passing of parameters. To mitigate memory leaks and simplify accounting, SOS provides a mechanism for requesting changes in data ownership when dynamically allocated memory is passed between modules. These design goals result in SOS choosing a more expressive scheduling mechanism at the cost of a more expensive scheduling loop. The API for messaging is SOS is shown in figure 5.

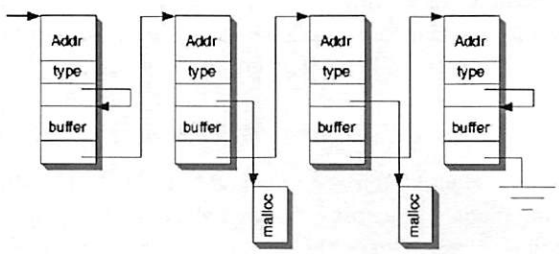


Figure 3—Memory Layout of An SOS Message Queue

Figure 3 provides an overview of how message headers are structured and queued. Message headers within a queue of a given priority form a simple linked list. The information included in message headers includes complete source and destination information, allowing SOS to directly insert incoming network messages into the messaging queue. Messages carry a pointer to a data payload used to transfer simple parameters and more complex data between modules. The SOS header provides an optimized solution to this common case of passing a few bytes of data between modules by including a small buffer in the message header that the data payload can be redirected to, without having to allocate a separate piece of memory (similar to the mbuf design). SOS message headers also include a series of flags to describe the priority of the message, identify incoming and outgoing radio messages, and describe how the SOS kernel should manage dynamic memory. These flags allow easy implementation of memory ownership transfer for a buffer moving through a stack of components, and freeing of memory on completion of a scheduled message.

SOS uses the high priority queue for time critical messages from ADC interrupts and a limited subset of timers needed by delay intolerant tasks. Priority queues have allowed SOS to minimize processing in interrupt contexts by writing interrupt handlers that quickly construct and schedule a high priority message and then drop out of the interrupt context. This reduces potential concurrency errors that can result from running in an interrupt context. Examples of such an errors include corruption of shared buffers that may be in use outside of the interrupt context, and stale interrupts resulting from operating with

Prototype	Description
int8_t post_short(sos_pid_t did, sos_pid_t sid, uint8_t type, uint8_t byte, uint16_t word, uint8_t flag)	Place a message on the queue to call function 'type' in module 'did' with data 'byte' and 'word'.
int8_t post_long(sos_pid_t did, sos_pid_t sid, uint8_t type, uint8_t len, void *data, uint8_t flag)	Place a message on the queue to call function 'type' in module 'did' with *data pointing to the data.

Table 5—Messaging API

Prototype	Description	Cycles
void *ker_malloc(uint16_t size, sos_pid_t id)	Allocate memory	69
void ker_free(void* ptr)	Free memory	85
int8_t ker_change_own(void *ptr, sos_pid_t id)	Change ownership	43
sos_pid_t ker_check_memory()	Validate memory (after a crash)	1175

Table 6—Cycles needed for dynamic memory tasks in SOS when running on an Atmel AVR microcontroller.

interrupts disabled.

3.3 Dynamic Memory

Due to reliability concerns and resource constraints, embedded operating systems for sensor nodes do not always support dynamic memory. Unfortunately, static memory allocation results in fixed length queues sized for worst case scenarios and complex program semantics for common tasks, such as passing a data buffer down a protocol stack. Dynamic memory in SOS addresses these problems. It also eliminates the need to resolve during module insertion what would otherwise be static references to module state.

SOS uses dynamic memory with a collection of bookkeeping annotations to provide a solution that is efficient and easy to debug. Dynamic memory in SOS uses simple best fit fixed-block memory allocation with three base block sizes. Most SOS memory allocations, including message headers, fit into the smallest block size. Larger block sizes are available for the few applications that need to move large continuous blocks of memory, such as module insertion. A linked list of free blocks for each block size provides constant time memory allocation and deallocation, reducing the overhead of using dynamic memory. Failed memory allocation returns a null pointer.

Queues and data structures in SOS dynamically grow and shrink at run time. The dynamic use and release of memory in SOS creates a system with effective temporal memory reuse and an ability to dynamically tune memory usage to specific environments and conditions. Self imposed hard memory usage limits prevent programing errors that could otherwise result in a module allocating all of the dynamic memory on a node.

Common memory functions and their clock cycle overheads are presented in table 6. All allocated memory is owned by some module on the node. This value is set during allocation and used by SOS to implement basic garbage collection and watch for suspect memory usage. Modules can transfer memory ownership to reflect data movement. Dynamic memory blocks are annotated with a small amount of data that is used to detect basic sequential memory overruns. These features are used for post-crash memory analysis[1] to identify suspect memory owners, such as a module owning a great deal of system memory or overflowed memory blocks. SOS also supports the use of a hardware watchdog timer used to force a soft boot of an unresponsive node and triggering the same post-crash memory analysis. Finally, memory block annotations enable garbage collection on module unload.

3.4 Miscellaneous

The SOS kernel includes a sensor API that helps to manage the interaction between sensor drivers and the modules that use them. This leads to more efficient usage of a node's ADC and sensing resources. Other standard system resources include timer multiplexing, UART libraries, and hardware management of the I2C bus.

The loosely coupled design used in SOS has resulted in a platform that is very portable. SOS currently has support for the Mica2 and MicaZ motes from Crossbow, and the XYZ Node from Yale. The most difficult portion of a port tends to be the techniques for writing to program memory while the SOS core is executing.

Limitations on module development result from the loose coupling of modules. An example of this is a 4KB size limitation for AVR module binaries, since relative jumps used in position independent code on the AVR architecture can only jump up to 4KB. Moreover, modules cannot refer to any global variables of the SOS kernel, as their locations of may not be available at compilation time.

4 Programming SOS Applications

Figure 4 contains a source code listing of the Sample_Send module of the Surge application[2]. The program uses a single *switch* structure to implement the message handler for the Sample_Send module. Using the standard C programing language reduces the learning curve of SOS while taking advantage of the many compilers, development environments, debuggers, and other tools designed for C. C also provides efficient execution needed to operate on resource limited 8-bit microcontrollers.

```
01 int8_t module(void *state, Message *msg) {
02   surge_state_t *s = (surge_state_t*)state;
03   switch (msg->type){
04     //! System Message - Initialize module
05     case MSG_INIT: {
06       char prototype[4] = {'C', 'v', 'v', '0'};
07       ker_timer_start(SURGE_MOD_PID, SURGE_TIMER_TID,
08             TIMER_REPEAT, INITIAL_TIMER_RATE);
09       s->get_hdr_size = (func_u8_t*)ker_get_handle
10             (TREE_ROUTING_PID, MOD_GET_HDR_SIZE,
11              prototype);
12       break;
13     }
14     //! System Message - Timer Timeout
15     case MSG_TIMER_TIMEOUT: {
16       MsgParam *param = (MsgParam*) (msg->data);
17       if (param->byte == SURGE_TIMER_TID) {
18         if (ker_sensor_get_data(SURGE_MOD_PID, PHOTO)
19             != SOS_OK)
20           return -EFAIL;
21       }
22       break;
23     }
24     //! System Message - Sensor Data Ready
25     case MSG_DATA_READY: {
26       //! Message Parameters
27       MsgParam* param = (MsgParam*) (msg->data);
28       uint8_t hdr_size;
29       uint8_t *pkt;
30       hdr_size = (*(s->get_hdr_size))();
31       if (hdr_size < 0) return -EINVAL;
32       pkt = (uint8_t*)ker_malloc
33           (hdr_size + sizeof(SurgeMsg), SURGE_MOD_PID);
34       s->smsg = (SurgeMsg*)(pkt + hdr_size);
35       if (s->smsg == NULL) return -EINVAL;
36       s->smsg->reading = param->word;
37       post_long(TREE_ROUTING_PID, SURGE_MOD_PID,
38             MSG_SEND_PACKET, length, (void*)pkt,
39             SOS_MSG_DYM_MANAGED);
40       break;
41     }
42     //! System Message - Evict Module
43     case MSG_FINAL: {
44       ker_timer_stop(SURGE_MOD_PID, SURGE_TIMER_TID);
45       ker_release_handle(s->get_hdr_size);
46       break;
47     }
48     default:
49     return -EINVAL;
40   }
51   return SOS_OK;
52 }
```

Figure 4—Surge Source Code

Line 2 shows the conversion of the generic state stored in and passed from the SOS kernel into the module's internal representation. As noted in section 3.1.1, the SOS kernel always stores a module's state to allow modules to be easily inserted at runtime.

An example of the *init* message handler appears on lines 5-13. These show the module requesting a periodic timer from the system and subscribing to the MOD_GET_HDR_SIZE function supplied by the tree routing module. This function pointer is used on line 30 to find the size of the header needed by the underlying routing layer. By using this function, changes to the underlying routing layer that modify the header size do not break the application. Lines 43-47 show the *final* message handler releasing the resources it had allocated: the kernel timer and the subscribed function. If these resources had not been explicitly released, the garbage collector in the SOS kernel would have soon released them. Good programing style is observed on lines 31 and 35 where potential errors are caught by the module and handled.

5 Evaluation

Our initial performance hypothesis was that SOS would perform at roughly the same level as TinyOS in terms of latency and energy usage, despite the greater level of functionality SOS provides (and the overhead necessary to support that functionality). We also hypothesized that SOS module insertion would be far less expensive in terms of energy than the comparable solution in TinyOS with Deluge. This section tests these hypotheses using a prototypical multihop tree builder and data collector called Surge. A baseline evaluation of the operating systems is also performed.

Benchmarking shows that application level performance of a Surge-like application on SOS is comparable to Surge on TinyOS and to Bombilla, an instantiation of the Maté virtual machine specifically for the Surge application. Initial testing of CPU active time in baseline evaluations of SOS and TinyOS showed a significant discrepancy between the two systems, motivating an in depth search into the causes of overhead in SOS. This search revealed simple optimizations that can be applied to improve both systems and bring their baseline performance to almost the same level. The process of remotely updating an application's functionality in SOS is more energy efficient than updates using Deluge in TinyOS, but less efficient than updates using Bombilla. While our hypotheses that SOS can more efficiently install updates than TinyOS with Deluge proves true, further analysis shows that this difference does not significantly impact the total energy usage over time and that the three systems have nearly identical energy profiles over long time scales. From these results, we argue that SOS effectively provides a flexible solution for sensor networking with energy usage functionally equivalent to other state of the art systems.

5.1 Methodology

We first describe the workings of the Surge application. Surge nodes periodically sample the light sensor and send that value to the base station over multi-hop wireless links. The route to the base station is determined by setting up a spanning tree; every node in the tree maintains only the address of its parent. Data generated at a node is always forwarded to the parent in the spanning tree. Surge nodes choose as their parent the neighbor with the least hop count to the base station and, in the event of a tie, the best estimated link quality. The estimate of the link quality is performed by periodically broadcasting beacon packets containing neighborhood information.

The Surge application in SOS is implemented with three modules: Sample_Send, Tree_Routing and PhotoSensor. For SOS, a blank SOS kernel was deployed on all motes. The Sample_Send, Tree_Routing and PhotoSensor modules were injected into the network and remotely installed on all the nodes. For TinyOS, the nodes were deployed with the Deluge GoldenImage. The Surge application was injected into the network using the Del-

Type of Code	Percentage
Kernel	0%
Mica2 Specific Drivers	21%
AVR Specific Drivers	36%

Table 7—Approximate number of lines of code (including comments) used from TinyOS in different parts of SOS.

uge protocol and remotely installed on all the nodes. For Maté, the Bombilla VM was installed on all the nodes. Bombilla VM implements the tree routing protocol natively and provides a high-level opcode, Send, for transferring data from the node to the base station via a spanning tree. Bytecode for periodically sampling and sending the data to the base station was injected into the network and installed on all relevant nodes.

In all versions of Surge, the routing protocol parameters were set as follows: sensor sampling happens every 8 seconds, parents are selected and route updates broadcast every 5 seconds, and link quality estimates are calculated every 25 seconds. Therefore, upon bootup, a node will have to wait at least 25 seconds before it performs the first link quality estimate. Since the parent is selected based on link quality, the expected latency for a node to discover its first parent in the network is at least 25 seconds.

The experiments in section 5.2 use the setup described above to examine application level performance of Surge. Section 5.3 examines the base SOS kernel and a simple TinyOS application to find the base CPU active time in the two operating systems when no application is present. The section continues with an examination of how the Surge application effects CPU active time. Section 5.4 finishes with a numerical examination of the update overhead in SOS, TinyOS and Maté. In an attempt to isolate operating system specific overhead, the analysis in section 5.4 does not use Deluge or the module distribution protocol currently used in SOS.

SOS reuses some of the TinyOS driver code for its Mica2 target and AVR support. Table 7 shows the approximate percentage of code reused from TinyOS in SOS. For these experiments SOS uses nearly direct ports of the TinyOS CC1000 radio stack, sensor drivers, and applications. This helps to isolate performance differences to within the kernel and module framework of SOS.

5.2 Macro Benchmarks

We first measure the time needed to form a routing tree and then measure the number of packets delivered from each node to the base station in a duration of 40 minutes. These tests act as a macro benchmark to help verify that Surge is running correctly on the three test platforms. Since all three systems are executing the same application and the CPU is not being heavily loaded, we expect differences in application performance to be small enough to be lost in the noise, as in fact they are.

For this experiment, Mica2 motes are deployed in a linear topology, regularly spaced at a distance of 4 feet from each other. The transmit power of the radio is set to -25dBm, the lowest possible transmit power, which results in a range of approximately 4 feet for our indoor laboratory environment.

Figure 5 shows the average time it took for a node to discover its first parent, averaged over 5 experiments. All three systems achieve latencies quite close to 25 seconds, the minimum latency possible given the Surge configuration parameters. The differences between the three systems are mainly due to their different boot routines. The source of the jitter results from the computation of the parent being done in a context which is often interrupted by packet reception.

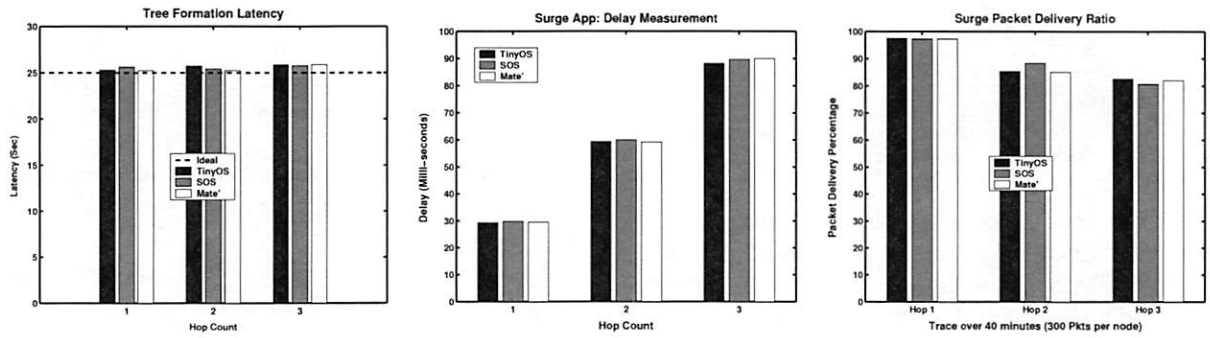
Figure 5 also shows the average time it takes to deliver the first packet from a node to the base station. Since all three systems implement a queue for sending packets to the radio, the sources of per-hop latency are queue service time, wireless medium access delay, and packet transmission time. Queue service time depends upon the amount of traffic at the node, but the only variation between the three systems is due to the differences in the protocols for code dissemination given the otherwise lightly loaded network. The medium access delay depends on the MAC protocol, but this is nearly identical in the three systems [18]. Finally, packet transmission time depends upon the packet size being transmitted; this is different by a handful of bytes in the three systems, causing a propagation delay on the order of 62.4 μs per bit [21]. The latency of packet delivery at every hop is almost identical in the three systems. The small variations that can be observed are introduced mainly due to the randomness of the channel access delay.

Finally, figure 5 shows the packet delivery ratio for the three systems when the network was deployed for 40 minutes. As expected, the application level performance of the three systems was nearly identical. The slight differences are due to the fluctuating wireless link quality and the different overheads introduced by the code update protocols.

These results verify our hypothesis that the application level performance of a typical sensor network application running at a low duty cycle in SOS is comparable to other systems such as TinyOS and Maté. The overheads introduced in SOS for supporting run time dynamic linking and message passing do not affect application performance.

5.3 CPU Overhead

We proceed by measuring CPU active time on the SOS and TinyOS operating systems without any application



(a) Surge Tree Formation Latency (b) Surge Forwarding Delay (c) Surge Packet Delivery Ratio

Figure 5—Macro Benchmark Comparison of Surge Application

activity to examine the base overhead in the two systems. Bombilla is not examined in these base measurements since the virtual machine is specific to the Surge application. The evaluation is finished by examining the CPU active time when running Surge on each of SOS, TinyOS, and Maté. In all three systems, the source code is instrumented to raise a GPIO pin in the microcontroller when the CPU is performing any active operation (executing a task or interrupt). A high speed data logger is used to capture the waveform generated by the GPIO pin and measure the active duration. All experiments are run for 10 intervals of 60 seconds using the same Mica2 motes. We expect both systems to have nearly identical base overheads since the SOS kernel should be nearly as efficient as the base TinyOS kernel.

To examine the base overhead in SOS we installed a blank SOS kernel onto a single mote and measured CPU active time to be $7.40\% \pm 0.02\%$. In contrast to this, running the closest TinyOS equivalent, TOSBase, resulted in a CPU active time of $4.76\% \pm 0.01\%$. This active time discrepancy came as a great surprise and prompted an evaluation into the cause of overhead in SOS.

Detailed profiling using the Avrora [23] simulator revealed that the overhead in SOS may be related to the SPI interrupt used to monitor the radio for incoming data. The SPI interrupt is triggered every $418\mu s$ in both SOS and TinyOS when the radio is not in a low power mode, as is the case in the above experiments. Micro-benchmarking showed that the SPI interrupt handler in SOS and TinyOS are almost identical. However upon exit from the SPI interrupt, control is transferred to the scheduler that checks for pending tasks. On a lightly loaded system the common case is to perform this check and go to sleep, since the scheduling queue is usually empty. Further micro-benchmarking revealed that in the common case of empty scheduling queues, the SOS scheduling loop takes 76 cycles compared to 25 cycles in TinyOS. This additional overhead acquired every $418\mu s$ accounts for the discrepancy in performance between SOS and TinyOS.

The SOS scheduling loop was then optimized to better handle this common case reducing the CPU active time to $4.52\% \pm 0.02\%$ and resulted in SOS outperforming TinyOS. For a fair comparison we modified the TinyOS scheduling loop and *post* operation [3] to use a similar scheduling mechanism, reducing the TinyOS CPU active time to $4.20\% \pm 0.02\%$. This difference is small enough that it can be accounted for by two additional push and pop instruction pairs in the SOS SPI handler that are introduced due to the slight modifications to the radio stack implementation.

The above active times are summarized in table 8. Of the most interest are the last two entries, which show that when both scheduling loops are optimized the base operation overhead of SOS and TinyOS are within a few percent, as expected.

Having verified that the base operating systems have very similar overheads, we continue by examining the CPU active time when Surge is run on SOS, TinyOS and Maté. Versions of both SOS and TinyOS with optimized schedulers are used for all three tests and Maté Bombilla runs on top of the optimized TinyOS core. Surge is loaded onto two nodes, and the active time is again measured by instrumenting the code to activate a GPIO pin when active and monitoring this pin with a high speed data logger. The experiment results are shown in Table 9. This comparison shows that this low duty cycle application has less effect on the base performance of SOS than for TinyOS. The cause of this larger increase for TinyOS is not yet fully understood. One hypothesis is that the initial baseline measurement between SOS and TinyOS do not accurately reflect the same base functionality.

5.4 Code Updates

We now present an evaluation of the energy needed for propagation and installation of the Surge application. We

OS Version	Percent Active Time
SOS Unoptimized	7.40% ± 0.02%
TinyOS Unoptimized	4.76% ± 0.01%
SOS Optimized	4.52% ± 0.02%
TinyOS Optimized	4.20% ± 0.02%

Table 8—CPU Active Time on Base Operating System

OS Version	Percent Active Time
SOS Optimized	4.64% ± 0.08%
TinyOS Optimized	4.58 ± 0.02%
Maté Bom.	5.13 ± 0.02%

Table 9—CPU Active Time With Surge

SOS Module Name	Code Size
Sample_Send	568 bytes
Tree_Routing	2242 bytes
Photo_Sensor	372 bytes
Energy (mJ)	2312.68
Latency (s)	46.6

Table 10—SOS Surge Remote Installation

begin by looking at installing Surge onto a completely blank node over a single hop in SOS. Surge on SOS consists of three modules: Sample_send, Tree_routing, and Photo_Sensor. Table 10 shows the size of the binary modules and overall energy consumption and the latency of the propagation and installation process. The energy consumption was measured by instrumenting the power supply to the Mica2 to sample the current consumed by the node.

We continue with an analytical analysis of the energy costs of performing similar code updates in our three benchmark systems—SOS, TinyOS with Deluge [10], and Maté Bombilla VM with Trickle [16]. It is important to separate operating system-related energy and latency effects from those of the code distribution protocol. Overall latency and energy consumption is mainly due to the time and energy spent in transmitting the updated code and storing and writing the received code to the program memory.

Communication energy depends upon the number of packets that need to be transferred in order to propagate the program image into the network; this number, in turn, is closely tied to the dissemination protocol. However, the number of application update bytes required to be transferred to update a node depends only on the architecture of the operating system. Therefore, to eliminate the differences introduced due to the different dissemination protocols used by SOS, TinyOS, and Bombilla, we consider the energy and latency of communication and the update process to be directly proportional to the number of application update bytes that need to be transferred and written to the program memory. The design of the actual distribution protocol used is orthogonal to the operating system design and SOS could use any of the existing code propagation approaches [22, 16]; it currently uses a custom publish/subscribe protocol similar to MOAP [22].

5.4.1 *Updating low level functionality*

To test updating low level functionality, we numerically evaluate the overhead of installing a new magnetometer sensor driver into a network of motes running the Surge application. The SOS operating system requires the distribution of a new magnetometer module, whose binary is 1316 bytes. The module is written into the program memory, one 256-byte page of the flash memory at a time. The cost of writing a page to the flash memory was measured to be 0.31 mJ, and 6 pages need to be written so the total energy consumption of writing the magnetometer driver module is 1.86 mJ.

System	Code Size (Bytes)	Write Cost (mJ/page)	Write Energy (mJ)
SOS	1316	0.31	1.86
TinyOS	30988	1.34	164.02
Maté VM	N/A	N/A	N/A

Table 11—Magnetometer Driver Update

System	Code Size (Bytes)	Write Cost (mJ/page)	Write Energy (mJ)
SOS	566	0.31	0.93
TinyOS	31006	1.34	164.02
Maté VM	17	0	0

Table 12—Surge Application Update

TinyOS with the Deluge distribution protocol requires the transfer of a new Surge application image with the new magnetometer driver. The total size of the Surge application with the new magnetometer driver is 30988 bytes. The new application image is first stored in external flash memory until it is completely received. The cost of writing and reading one page of the external flash is 1.03 mJ [21]. Thereafter, the new image is copied from the external flash to the internal flash memory. This makes the total cost of installing a page of code into the program memory 1.34 mJ, and the total energy consumption for writing the updated Surge application 164.02 mJ.

Lastly, the Maté Bombilla VM does not support any mechanism for updating low level functionality such as the magnetometer driver. These results are summarized in table 11.

5.4.2 *Updating application functionality*

We examine updates to applications by numerically evaluating the overhead of updating the functionality of the Surge application. The modified Surge application samples periodically, but transmits the value to a base station only if it exceeds a threshold. The SOS operating system requires the distribution of a new Sample_Send module with the updated functionality. As before, TinyOS/Deluge requires the transfer of a new Surge application image with the modified functionality. However, the Maté Bombilla VM requires just a bytecode up-

date. The total size of the bytecode was only 17 bytes, and since it is installed in the SRAM, it has no extra cost over the running cost of the CPU. The results of an analysis similar to that in section 5.4.1 is presented in table 12.

SOS offers more flexibility than Maté; operations such as driver updates not already encoded in Maté's high-level bytecode cannot be accomplished. However, Maté's code updates are an order of magnitude less expensive than SOS's. TinyOS with Deluge picks the extreme end of the flexibility/cost trade off curve by offering the greatest update flexibility at the highest cost of code update. With a Deluge like mechanism, it is possible to upgrade the core kernel components, which is not possible using only modules in SOS; but in SOS and Maté, the state in the node is preserved after code update while it is lost in TinyOS due to a complete reboot of the system.

The above analysis does not consider how differential updates described in section 2.3 could impact static and dynamic systems. There is potential for differential updates to significantly decrease the amount of data that needs to be transmitted to a node for both systems. It is also unclear how efficiently a differential update, which may require reconstructing the system image in external flash, can be implemented. Differential updates are an area of research that both SOS and TinyOS could benefit from in the future.

5.4.3 *Analytical analysis of update energy usage*

The previous evaluation of CPU utilization and update costs in SOS, TinyOS, and Maté prompts asking how significant those factors are in the total mote energy expenditure over time. Application energy usage on a mote can be divided up into two primary sources: the energy used to install or update the application, and the power used during application execution multiplied by application execution time. An interesting point of comparison is to find the time, if any, when the total energy usage of two systems running the same application becomes equal. Starting with an update to a node, the total energy consumption of the node is found using:

$$E_{total} = E_{update} + P_{average} \times T_{live} \quad (1)$$

$P_{average}$ denotes the power consumption of the mote averaged over application execution duration and the sleep duration (due to duty cycling). T_{live} is the time that the node has been alive since it was updated. E_{update} is the energy used to update or install the application.

The average power consumption during application execution depends upon the active time of the application, duty cycle of operation, and the power consumption of the hardware platform in various power modes. The Mica2 power consumption in various modes was obtained from [21] and is summarized in table 13. We first compute P_{awake}, the average power consumption of the Surge application assuming a 100% duty cycle i.e. the system is operating continuously without sleep:

Mode	Active (mW)	Idle (mW)	Sleep (mW)
Mica2 Power	47.1	29.1	0.31
Duty Cycle(%)	**TinyOS (mW)**	**SOS (mW)**	**Maté (mW)**
100	29.92	29.94	30.02
10	3.271	3.272	3.281
1	0.6057	0.6058	0.6067

Table 13—Average Power Consumption of Mica2 Executing Surge

$$P_{awake} = P_{active} \times \frac{T_{active}}{T_{active} + T_{idle}} + P_{idle} \times \frac{T_{idle}}{T_{active} + T_{idle}}$$

Table 9 provides the ratio $\frac{T_{active}}{T_{active}+T_{idle}}$, which can be used to find $\frac{T_{idle}}{T_{active}+T_{idle}}$. When the system is operated at lower duty cycles, the average power consumption is given by:

$$P_{average} = P_{awake} \times DutyCycle + P_{sleep} \times (1 - DutyCycle)$$

Table 13 summarizes the average power consumption of the Surge application at various duty cycles.

Now we compute the energy used during the Surge application installation, E_{update}, which is the sum of the energy consumed in receiving the application over the radio and subsequently storing it on the platform. For the Mica2 Mote, the energy to receive a byte is 0.02 mJ/byte-[21]. Using table 12 we can compute the energy required to update the Surge application on SOS assuming that communications energy has a one to one correlation with application bytes transmitted:

$$E_{update}(SOS) = 0.02\frac{mJ}{byte} \times 566bytes + 0.93mJ = 12.25mJ$$

Similar computations can also be performed for TinyOS and Maté using the numbers in table 12.

Using the numbers computed thus far, it is possible to understand the total energy consumption of a system, E_{total}, as a function of the elapsed time T_{live}. From equation 1, it can be seen that energy consumption is linear with the slope being equal to $P_{average}$ with an initial offset equal to E_{update}. The average power consumption for each of the three systems examined is very similar and results in nearly identical total energy usage over time, despite the initially significantly different update costs. For example, by solving the linear equations for the 10% duty cycle, it is easy to see that TinyOS becomes more energy efficient than SOS after about 9 days. Similarly, for a 10% duty cycle, SOS becomes more energy efficient than Maté after about 22 minutes. But looking at the absolute difference in energy consumed after 30 days at a 10% duty cylcle, SOS has consumes less than 2 J more energy than TinyOS and Maté is only about 23 J beyond SOS. This analysis reveals that, contrary to our

initial intuition, differences in both CPU utilization and update costs are not significant in the total energy consumption of the node over time.

Note that this analysis is only for a single application, Surge, running under a given set of conditions. Different applications and evolving hardware technology will change the results of the above analysis. Applications with very high CPU utilization can magnify the effects of different CPU utilization on $P_{average}$ resulting in total energy variances of 5% to 10% on current mote hardware. Similarly, improvements to duty cycling that reduce node idle time can magnify the effects of CPU utilization on $P_{average}$. Hardware innovations that reduce peripheral energy usage, especially when the CPU is idle, will also lead to a noticeable discrepancy in total energy consumption. Only as the above improvements to systems are made, leading to more energy efficient execution environments, will energy savings from efficient updates become significant.

5.5 Discussion

This analysis examines energy usage, which provides a quantifiable comparison of these systems. This section also shows that these differences are not significant when taking into account average power consumption resulting from the different power modes of current mote technology. Choosing between operating systems for these systems should be driven more by the features provided by the underlying system, such as flexibility of online updates in SOS or full system static analysis in TinyOS, rather than total system energy consumption. As software and hardware technologies advance, the effects of differing CPU utilization and update costs between systems will play a more significant role in choosing a system.

The results of the experiments performed in this section validate the following claims about SOS. First, the architectural features presented in section 3 position SOS at a middle ground between TinyOS and Maté in terms of flexibility. Second, the application level performance of the SOS kernel is comparable to TinyOS and Maté for common sensor network applications that do not stress the CPU. Third, code updates in SOS consume less energy than similar updates in TinyOS, and more energy than similar updates in Maté. Fourth, despite different update costs and CPU active times, the total energy usage in SOS is nearly identical to that in TinyOS and Maté.

6 Conclusion

SOS is motivated by the value of maintaining modularity through application development and into system deployment, and of creating higher-level kernel interfaces that support general-purpose OS semantics. The architecture of SOS reflects these motivations. The kernel's message passing mechanism and support for dynamically allocated memory make it possible for independently-created binary modules to interact. To improve the performance of the system and to provide a simple programming interface, SOS also lets modules interact through function calls. The dynamic nature of SOS limits static safety analysis, so SOS provides mechanisms for run-time type checking of function calls to preserve system integrity. Moreover, the SOS kernel performs primitive garbage collection of dynamic memory to improve robustness. Fine grain energy usage of SOS, TinyOS and the Maté Bombilla virtual machine are dependent on the frequency of updates and stressing of the CPU required for a specific deployment. However, analysis of overall energy consumed by each of the three systems at various duty cycles shows nearly identical energy usage for extended deployments. Thus, to choose between SOS and another system, it is important for developers to consider the benefits of static or dynamic deployments independently of energy usage.

SOS is a young project under active development and several research challenges remain. One critical challenge confronting SOS is to develop techniques that protect the system against incorrect module operation and help facilitate system consistency. The stub function system, typed entry points, and memory tracking protect against common bugs, but in the absence of any form of memory protection it remains possible for a module to corrupt data structures of the kernel and other modules. Techniques to provide better module isolation in memory and identify modules that damage the kernel are currently being explored. The operation of SOS is based on the notion of co-operative scheduling, but an incorrectly composed module can utilize all of the CPU. We are exploring more effective watchdog mechanisms to diagnose this behavior and take corrective actions. Since binary modules are distributed by resource constrained nodes over a wireless ad-hoc network, SOS motivates more research in energy efficient protocols for binary code dissemination. Finally, SOS stands to benefit greatly from new techniques and optimizations that specialize in improving system performance and resource utilization for modular systems.

By combining a kernel that provides commonly used base services with loosely-coupled dynamic modules that interact to form applications, SOS provides a more general purpose sensor network operating system and supports efficient changes to the software on deployed nodes.

We use the SOS operating system in our research and in the classroom, and support users at other sites. The system is freely downloadable; code and documentation are available at:
http://nesl.ee.ucla.edu/projects/sos/.

ACKNOWLEDGEMENTS

We gratefully acknowledge the anonymous reviewers and our shepherd, David Culler. Thanks goes out the XYZ developers and other users of SOS who have provided feedback and help to make better system. This material is based on research funded in part by ONR through the AINS program and from the Center for Embedded Network Sensing.

REFERENCES

[1] ABRACH, H., BHATTI, S., CARLSON, J., DAI, H., ROSE, J., SHETH, A., SHUCKER, B., DENG, J., AND HAN, R. Mantis: system support for multimodal networks of in-situ sensors. In *Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications* (2003), ACM Press, pp. 50–59.

[2] BERSHAD, B. N., SAVAGE, S., PARDYAK, P., SIRER, E. G., FIUCZYNSKI, M., BECKER, D., EGGERS, S., AND CHAMBERS, C. Extensibility, safety and performance in the SPIN operating system. In *15th Symposium on Operating Systems Principles* (Copper Mountain, Colorado, 1995), pp. 267–284.

[3] BOULIS, A., HAN, C.-C., AND SRIVASTAVA, M. B. Design and implementation of a framework for efficient and programmable sensor networks. In *Proceedings of the First International Conference on Mobile Systems, Applications, and Services* (2003), ACM Press, pp. 187–200.

[4] CROSSBOW TECHNOLOGY, INC. *Mote In-Network Programming User Reference*, 2003.

[5] DUNKELS, A., GRÖNVALL, B., AND VOIGT, T. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Proceedings of the First IEEE Workshop on Embedded Networked Sensors* (2004).

[6] ENGLER, D. R., KAASHOEK, M. F., AND O'TOOLE, J. Exokernel: An operating system architecture for application-level resource management. In *Symposium on Operating Systems Principles* (1995), pp. 251–266.

[7] FOK, C., ROMAN, G.-C., AND LU, C. Rapid development and flexible deployment of adaptive wireless sensor network applications. Tech. Rep. WUCSE-04-59, Washington University, Department of Computer Science and Engineering, St. Louis, 2004.

[8] GAY, D., LEVIS, P., VON BEHREN, R., WELSH, M., BREWER, E., AND CULLER, D. The nesc language: A holistic approach to networked embedded systems. In *Proceedings of Programming Language Design and Implementation* (2003).

[9] HILL, J., SZEWCZYK, R., WOO, A., HOLLAR, S., CULLER, D., AND PISTER, K. System architecture directions for networked sensors. In *Proceedings of the ninth international conference on Architectural support for programming languages and operating systems* (2000), ACM Press, pp. 93–104.

[10] HUI, J. W., AND CULLER, D. The dynamic behavior of a data dissemination protocol for network programming at scale. In *Proceedings of the second international conference on Embedded networked sensor systems* (2004), ACM Press.

[11] JEONG, J., AND CULLER, D. Incremental network programming for wireless sensors. In *Proceedings of the First IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks IEEE SECON* (2004).

[12] JUANG, P., OKI, H., WANG, Y., MARTONOSI, M., PEH, L.-S., AND RUBENSTEIN, D. Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with ZebraNet. In *Proc. 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X)* (San Jose, California, Oct. 2002).

[13] KAISER, W., POTTIE, G., SRIVASTAVA, M., SUKHATME, G. S., VILLASENOR, J., AND ESTRIN, D. Networked Infomechanical Systems (NIMS) for ambient intelligence.

[14] LEVIS, P., AND CULLER, D. Mate: A tiny virtual machine for sensor networks. In *International Conference on Architectural Support for Programming Languages and Operating Systems, San Jose, CA, USA* (Oct. 2002).

[15] LEVIS, P., MADDEN, S., GAY, D., POLASTRE, J., SZEWCZYK, R., WOO, A., BREWER, E., AND CULLER, D. The emergence of networking abstractions and techniques in tinyos. In *Proceedings of the First Symposium on Networked Systems Design and Implementation* (2004), USENIX Association, pp. 1–14.

[16] LEVIS, P., PATEL, N., CULLER, D., AND SHENKER, S. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *Proceedings of the First Symposium on Networked Systems Design and Implementation* (2004), USENIX Association, pp. 15–28.

[17] LIU, T., AND MARTONOSI, M. Impala: a middleware system for managing autonomic, parallel sensor systems. In *Proceedings of the ninth ACM SIGPLAN symposium on Principles and practice of parallel programming* (2003), ACM Press, pp. 107–118.

[18] POLASTRE, J., HILL, J., AND CULLER, D. Versatile low power media access for wireless sensor networks. In *Second ACM Conference on Embedded Networked Sensor Systems* (2004).

[19] RASHID, R., JULIN, D., ORR, D., SANZI, R., BARON, R., FORIN, A., GOLUB, D., AND JONES, M. B. Mach: a system software kernel. In *Proceedings of the 1989 IEEE International Conference, COMPCON* (San Francisco, CA, USA, 1989), IEEE Comput. Soc. Press, pp. 176–178.

[20] REIJERS, N., AND LANGENDOEN, K. Efficient code distribution in wireless sensor networks. In *Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications* (2003), ACM Press, pp. 60–67.

[21] SHNAYDER, V., HEMPSTEAD, M., RONG CHEN, B., AND MATT WELSH, H. Powertossim: Efficient power simulation for tinyos applications. In *Sensor Networks. In Proc. of ACM SenSys 2003.* (2003).

[22] STATHOPOULOS, T., HEIDEMANN, J., AND ESTRIN, D. A remote code update mechanism for wireless sensor networks. Tech. Rep. CENS-TR-30, University of California, Los Angeles, Center for Embedded Networked Computing, November 2003.

[23] TITZER, B. L., PALSBERG, J., AND LEE, D. K. Avrora: Scalable sensor network simulation with precise timing. In *Fourth International Conference on Information Processing in Sensor Networks* (2005).

NOTES

[1] Soft reboot of many microcontrollers, including the Atmel AVR, preserves on chip memory.

[2] This source code is trimmed for clarity. Complete source code can be downloaded separately.

[3] Patch is being submitted to the TinyOS developers should they wish to optimize for the case of a lightly loaded radio that is not in a low power mode.

A Relative Positioning System for Co-located Mobile Devices

Mike Hazas, Christian Kray, Hans Gellersen, Henoc Agbota, Gerd Kortuem
Computing Department, Lancaster University, United Kingdom

Albert Krohn
TecO, University of Karlsruhe, Germany

Abstract

If a mobile computing device knows how it is positioned and oriented in relation to other devices nearby, then it can provide enhanced support for multi-device and multi-user interactions. Existing systems that provide position information to mobile computers are reliant on externally deployed infrastructure, such as beacons or sensors in the environment. We introduce the *Relate* system, which provides fine-grained relative position information to co-located devices on the basis of peer-to-peer sensing, thus overcoming dependence on any external infrastructure. The system is realised as a hardware/software plug-in, using ultrasound for peer-to-peer sensing, USB to interface with standard mobile devices, and data abstraction and inferencing to map sensor data to a spatial model that maintains both quantitative and qualitative relationships. We present a set of services and applications to demonstrate the utility of the system. We report experimental results on the accuracy of the relative position and orientation estimates, and other aspects of system performance.

1 Introduction

The significance of location information to mobile computers has been established through a large body of research over the last fifteen years. In many cases, it is not the *absolute* location of a device that becomes used for spatially-aware applications, but rather their *relative* position with respect to other devices. Relative position information can be used to facilitate easy configuration and connection of devices that come into proximity, and to enhance interaction in situations where there are many devices. There are benefits for single users having multiple mobile devices, as well as for multiple co-located users each having their own device. Applications span from planned multi-device or multi-user tasks to spontaneous interaction and collaboration. Some specific application examples include synchronisation of multiple devices through meaningful spatial arrangement [10, 20], and configuring connectivity and interaction between devices being used by collaborating mobile users [2, 24].

Location information provided by existing systems is generally too coarse-grained to be useful for devices that are already co-located. For example, GPS or WiFi signals can be used to infer a general sense of nearness but are not sufficient for fine-grained modelling of spatial relationships; to support operations and interactions on the scale of personal devices, an accuracy on the order of ten centimetres is ideal. Certain indoor location systems are capable of providing such fine-grained location and orientation information sufficient for elaborate relative positioning tasks [1, 4, 15, 18]. Systems have also been developed specifically for sensing co-located devices, especially in physical user interfaces. These have used a number of techniques, including computer vision [20, 21, 25, 26], physical contact or weight [10, 23], optical mouse tracking [5], and short-range electromagnetic signals [12, 17]. Many of these systems have a high cost due to their sensor density, specialised transducers, or installation and calibration effort. Moreover, all of these systems require static, pre-installed infrastructure, restricting applications to *specifically instrumented, indoor environments*. For mobile users, systems requiring instrumentation of the environment are inhibiting.

This paper describes the Relate system which we have designed to enable co-located mobile computing devices to directly establish their spatial relationships, without need for infrastructure in their environment. The system is based on wireless sensor devices implemented as USB peripherals (*dongles*) that can be readily used to extend mobile computers (*hosts*) with peer-to-peer sensing. The dongles are designed to perform best when the devices are positioned approximately in the same plane. Mobile computers augmented with Relate that are within an interaction range of about two meters discover each other to form a dynamic Relate network. This network is in-

dependent of any other networks that the involved hosts may have available or established between them. The Relate network is used to coordinate and collect range and angle-of-arrival measurements using ultrasonic signals between the dongles, and also to share information such as host and user names. Each mobile host in a Relate network uses this information to independently establish and maintain a spatial model of the network.

The spatial model maintained by mobile hosts is a real-time representation of the spatial configuration of all hosts in a Relate network. The model represents hosts in a two-dimensional coordinate system relative to the local host device. It accounts for changes such as arrival and departure of hosts, and it provides both quantitative and qualitative information. This includes fine-grained position and orientation of the devices in a relative coordinate space, host and user names, and spatial relationships such as `left_of`, `right_of`, `approaching`, and `moving_away`. Applications can make direct use of the spatial model through query and event mechanisms but they can also use it indirectly through spatially-aware services, which provide functionalities such as sending a message to all devices within half a metre.

The Relate system architecture is laid out in section 2, and each of its components, from the sensor layer through the spatial modelling layer to application services are described in detail in sections 3 to 5. An experimental evaluation of the system is presented in section 6.

2 System Architecture

The Relate system architecture is depicted in figure 1. It provides relative positioning capability to mobile computers based on a layered architecture, with each layer providing a clear set of functions to the layer above. The four layers from bottom to top are:

1. The **sensor layer** performs distance and angle-of-arrival measurements between dongles, and collects measurements from participating dongles as an input stream to the layer above.

2. The **model layer** feeds data provided by the sensor layer through a data processing pipeline, and maps sensor data to a representation of the spatial configuration of hosts. The pipeline involves computation of relative coordinates, abstraction of qualitative information, and inference of information over time.

3. The **service layer** provides access to the spatial model. This happens directly through event and query mechanisms, and indirectly through spatially-aware communication services that implicitly use the model.

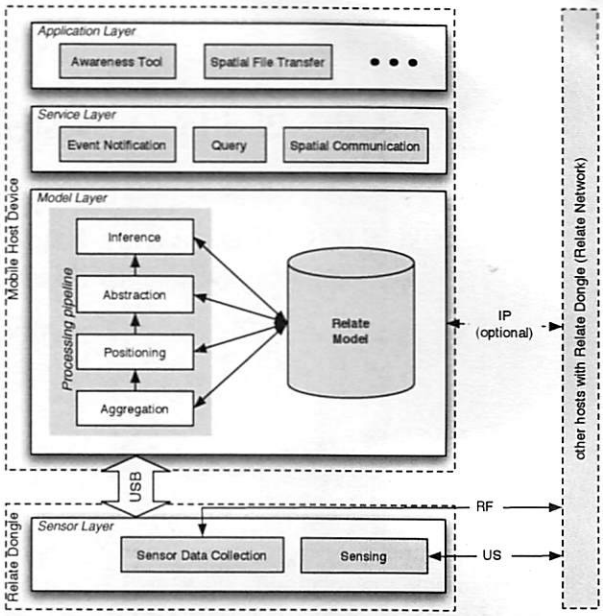


Figure 1: The Relate System Architecture.

4. In the **application layer**, user-level programs utilise services in the layer below to implement spatially-aware behaviour.

As mentioned above, the functional architecture maps to a physical structure of mobile hosts augmented with sensing dongles. The Relate dongles cooperatively implement the functionality of the sensor layer. All other layers are implemented in the mobile host which is a mobile computing device with a USB interface. Note that these layers, unlike the sensor layer, do not necessarily require cooperation. Each host maintains a Relate model computed independently of other hosts, and supports services and applications on top of its model. Spatial awareness is obviously of interest for applications that involve communication or collaboration between mobile hosts, however applications can also be stand-alone. An example for a stand-alone application is a visualisation of the Relate network which a host can generate from its model without cooperation above the sensor layer.

A Relate system involves various forms of inter-device communication. A host is connected to its dongle via USB. It is assumed that this connection is stable during a single session, although there is no mandate that a host be paired repeatedly with any particular dongle. Inter-dongle operation in the sensor layer is broadcast-based and involves two separate channels. An RF channel is used for coordination of peer-to-peer sensing and sensor data exchange, and the actual sensing is performed using an ultrasonic channel. The dongle network is ad hoc and dynamic. Neither dongles nor hosts need a priori knowl-

edge to establish communication, and the participating devices may change due to user mobility.

A Relate system may optionally involve host-to-host communication via IP to support spatially-aware communication and collaboration. They can also use IP connectivity for sharing of sensor data in addition to the sensor data propagation provided by the dongle network, to improve overall availability of sensor data for spatial modelling. If hosts do not have an IP connection however, they can only support stand-alone services and applications.

3 Relate Sensor Layer

As explained earlier, each dongle is an ad hoc wireless sensing add-on to a mobile host for which it collects data. This involves a custom hardware device and protocols for ad hoc networking and distributed sensing.

3.1 Relate Dongles

A Relate USB dongle is shown in figure 2. It is composed of a circuitboard with a microcontroller and RF module (a Smart-Its Particle[1]), and a separate circuitboard with sensors and a USB interface. The dongle casing is about 5.5 × 3.5 × 1.5 cm in size, and has a standard A-type USB connector on one side. The other three sides of the dongle each have a one-centimetre ultrasonic transducer, arranged to cover the space left, right and in front of the USB port on the host device.[2] As mentioned above, the dongles perform best when they lie approximately in the same plane.

Figure 3 depicts the hardware architecture of the dongle, which is based around a PIC18F452 microcontroller. Unlike many previous systems utilising ultrasonic ranging for location sensing, the interface circuitry to the transducers in the Relate dongle is *bidirectional*. To emit ultrasound, the PIC's digital output pins drive the transducers through buffering transistors. To detect ultrasonic pulses, the transistor buffers are used to put the transducers into a tri-state mode, and the PIC samples the signals present on the transducer terminals.

The PIC is also connected to an RFM radio transceiver operating in the 868.35 MHz ISM band. The transceiver provides wireless synchronisation and communication between the dongles in the Relate network. Finally, the dongle's PIC communicates with its host via an FT232BM, which is an RS232-to-USB bridge chip. Host device drivers for the bridge are readily available for Windows, Linux, and Macintosh operating systems. Upon connection of the bridge to the USB bus of the host, the drivers create a virtual serial port which provides the interface to services and applications.

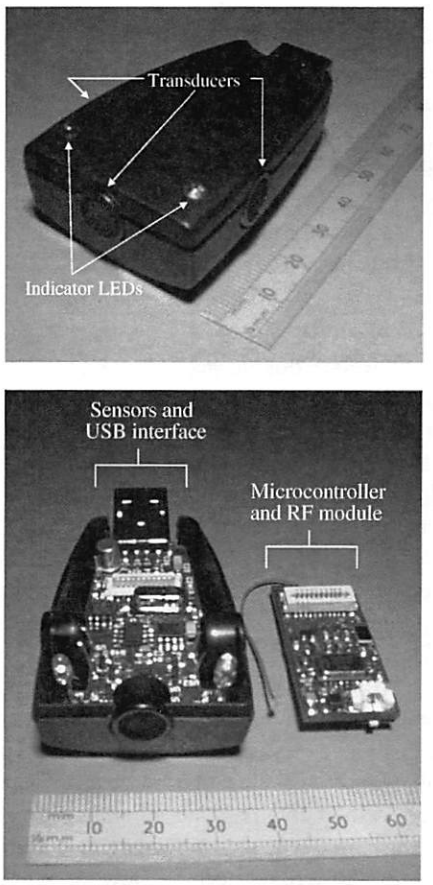


Figure 2: The Relate hardware is packaged as a USB dongle (top). The microcontroller and sensing components are on separate circuitboards (bottom) which snap together to fit inside the dongle casing.

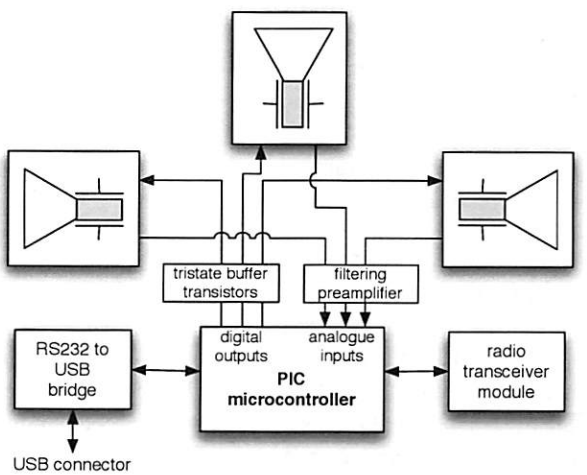


Figure 3: The Relate dongle integrates a wireless embedded computing core with ultrasonic transducers and a USB interface.

3.2 Dongle Network

The dongle devices communicate over a peer-to-peer network with random access. The network provides precise time synchronisation to all nodes (less than $4\mu s$) and utilises a slotted TDMA collision avoidance protocol with 13 ms packet length and 64 byte payload per packet. Relate-specific protocol functions implemented on top of this network include Relate network discovery and management, coordination of ultrasonic sensing, and collection of sensor data.

When a Relate dongle gets access to the network it utilises a number of time slots for data transmission and for ultrasonic signalling. Minimally, seven slots are used: two for RF communication, three for ultrasonic signalling, one for USB communication, and a final one for preparation of the next transmission cycle (see figure 4). The first RF packet is used for transmission of observed network state as a table of device IDs each with a timestamp of their last sighting. Receiving nodes use this table to update their own view of the network. This is followed by a packet that communicates range and angle-of-arrival measurements that have been taken since the dongle last had access to the network.[3] Receiving dongles collect this to provide their hosts with measurements from across the network. The remaining time slots are reserved for ultrasonic sensing and USB communication with the host, described further below.

When a Relate dongle first connects to a network it interrupts the protocol cycle described above to send a time request packet. This prompts devices on the network to set a flag to use their next transmission cycle to include real-time clock information in their network state packet. By this means, all dongles are synchronised to a global "dongle network time" to keep track of when other dongles have last been sighted on the network. This information is required to maintain a shared and consistent understanding of how many nodes are present on the network, as the protocol adapts to network size.

The network state table is shared in a multi-hop fashion. For example, if dongle A does not have direct RF communication to dongle B, it still keeps a record of when dongle B was last seen, based on the network state packets received from the other dongles. Dongle A also broadcasts this information in its own network state packets. This multi-hop network state information is used to implement fair access, with each device backing off for the number of slots required for all other dongles to have a complete transmission cycle.

Another function of the Relate protocol is to propagate host information through the dongle network. Each dongle receives information from its host device once it becomes connected to it. This includes host name, user name and IP address. This information is sent at a certain interval (in our current implementation every five seconds) as a host information packet over the dongle network. Receiving dongles pass this information on to their hosts. Hosts store the information about one another in their model layer, and use the transmitted IP addresses to test availability of an IP connection between one another.

3.3 Ultrasonic Sensing

After a Relate dongle has obtained network access and used two time slots to transmit RF packets, the following three time slots are devoted to ultrasonic sensing. In each time slot, the transmitting device emits ultrasound from all its transducers simultaneously. The other dongles use the two RF packets as a trigger to listen for ultrasonic pulses. Because of the analogue sampling limitations of the PIC18F, the dongles can only monitor one of their transducers at a time. Therefore three consecutive 13 ms time slots are used for emission of ranging pulses, in order for the receiving dongles to gather data using all three of their transducers.

The receiving dongles use data only from transducers on which they detect ultrasonic pulses of sufficient strength, and measure the peak signal values and the times-of-flight of the ultrasonic pulses sent by the transmitting device. The smallest time-of-flight is then used to calculate a range estimate. In addition, an angle-of-arrival estimate is derived using the known orientation of transducers on the dongle and calculated based on the relative spread of peak signal values measured across these transducers.

3.4 Data Collection

Relate devices collect sensor data as six-tuples including the IDs of the transmitting dongle and receiving dongle, the range and angle-of-arrival estimates, the number of transducers at which a sufficiently strong pulse was detected, and a timestamp. Each dongle maintains a buffer into which it writes its own sensor observations made when another dongle emits ultrasonic pulses, as well as sensor data received from other dongles via radio.

All dongles use a dedicated time slot after ultrasonic sensing to transmit sensor data from their buffer over the USB link to their host. At any time, the content of the buffers across the dongle network will vary to an extent due to the delay between local calculation of sensor data and its propagation to other dongles in the network. As a result, the data received by hosts in the Relate network will include measurements taken by other dongles in the sensor network (and not only by the directly attached dongle). However, the data will not be absolutely consistent across hosts at any particular time, as local sensor readings are available ahead of remote ones.

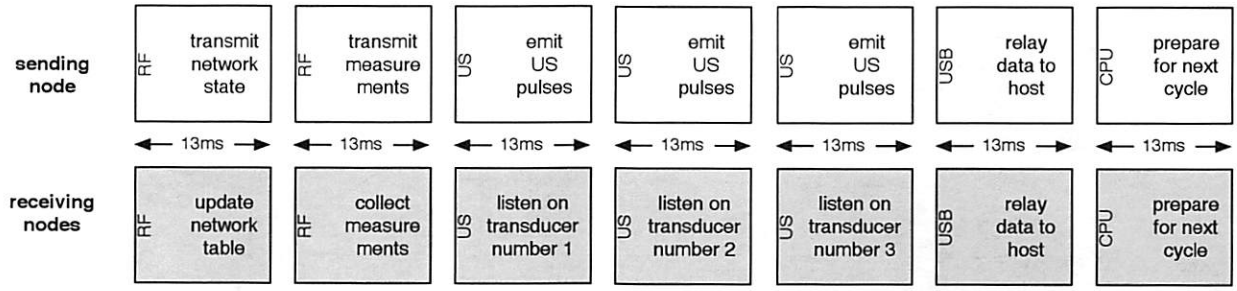


Figure 4: Relate dongles use a slotted protocol for data transmission over RF and ultrasonic sensing.

4 Spatial Modelling Layer

Relate hosts use the sensor data provided by the dongle network to generate and maintain a spatial model as a real-time representation of the spatial configuration of all devices in a Relate network. This involves a data processing pipeline with four consecutive stages for cyclic updates of the model.

4.1 Relate Model

The Relate model is a list of labelled directed graphs. Each graph represents the spatial arrangement of devices in a Relate network at a particular point in time. The list represents an ordered history of spatial information, with the most recent graph in the list representing the current arrangement of devices. A graph may be incomplete as it is dependent on information provided from the sensor layer. The nodes in the graph represent devices and the edges indicate spatial relationships between devices. Both nodes and edges are labelled with attribute-value pairs to capture properties of devices and their relations. The attributes of devices are summarised in table 1.

The relationships between devices fall into three categories. First, relationships between one device and another can be derived from raw **sensor data**, and expressed as a six-tuple data object: transmitting and receiving dongle IDs, range, angle-of-arrival, number of receiving transducers, and time stamp. Each pair of devices can be associated with up to two measurements, as measurements are directed. Second **quantitative relationships** can be expressed as the distance between devices, and bearing of one device with respect to another in the local coordinate system. Third, **qualitative relationships** describe the relative spatial arrangement of one device with respect to another (`left_of`, `in_front_of`, `right_of`), and of relative movement (`approaching`, `moving_away`).

The Relate model is continuously updated by processing the stream of sensor data generated by the sensor layer below. The sensor data is processed in a staged pipeline with each stage computing certain information for the model. Each processing cycle through this pipeline results in a new graph that is indexed and timestamped, and added to the list. Previously computed graphs are maintained in the list to facilitate reasoning about spatiotemporal information such as relative device trajectories. Over time, graphs may become invalid for temporal reasoning and removed.

Table 1: Device attributes captured in the Relate model.

Attribute	Description

`host_name`
`host_type`
`user_name`
`IP_address`
	Host information: the type refers to class of device, e.g. notebook or PDA

`dongle_ID` `dongle_orient`	Dongle information: ID of the host's dongle, and dongle orientation in relation to its host
`position` `orientation`	(x, y) coordinates and orientation of the device in a relative coordinate system with the local device at the origin

For illustration of the graph representation, an example is shown in figure 5. Hosts with dongles are represented as nodes and their relationships as a set of directed or bidirectional edges. The lower part of the figure depicts several quantitative and qualitative relations between node 5 and node 1. The two topmost edges represent measurements taken by node 1 observing node 5 and vice versa. As the example shows, range estimates can vary due to measurement error. Other relationships shown include distance as well as qualitative relations, all resulting from processing of raw sensor data.

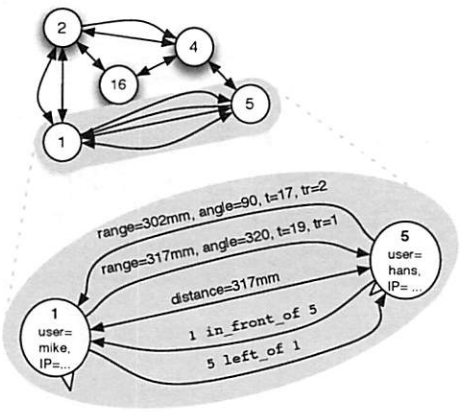


Figure 5: An example for the graph representation in the Relate spatial model: hosts with dongles are represented as nodes and their relationships as a set of directed or bidirectional edges.

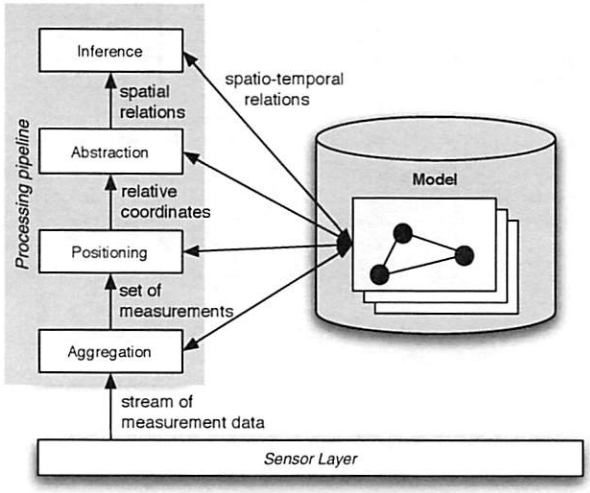


Figure 6: Sensor data is processed in staged pipeline to compute Relate model updates.

4.2 Data Processing Pipeline

The processing pipeline for the spatial model consists of four stages, with the first one operating on the data provided by the sensor layer, and the following ones each operating on the output of the previous stage. The pipeline generates and maintains the Relate model as summarised in figure 6.

4.2.1 Aggregation

The initial pipeline stage generates a new graph at the start of each processing cycle and adds it to the Relate model's list, indexed with the next increment. It continuously reads sensor data packets provided by the sensor layer from the USB interface. Each packet represents a measurement described as a six-tuple. The data it contains is extracted and added to the model in the form of new device relationships. Sensor data is aggregated for the current graph until a specified condition is met that triggers the next processing stage in the pipeline, and a new cycle for sensor data aggregation. This condition can be time-based (e.g. "trigger every 500 ms"), data-driven (e.g. "trigger every 100 data packets"), or a combination of both ("trigger every second or when 50 new data packets have been aggregated"). When this condition is met, then the current graph is completed with recent `sensor_data` objects from the previous graphs to ensure that the newly inserted graph contains a complete set of sensor measurements irrespective of the duration of an aggregation cycle or the number of newly gathered sensor data packets.

4.2.2 Positioning

The second stage operates on the `sensor_data` objects in the Relate graph to compute quantitative spatial attributes and relations, i.e. device relative positions and orientations. The range and angle-of-arrival measurements can be used to arrive at a solution for the relative positions and orientations of the devices using a system of equations. More specifically, the range d_{ij} between device i, located at (x_i, y_i), and device j, located at (x_j, y_j), can be defined as follows:

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}. \quad (1)$$

Also, the reception angle ψ_{ij} at device i with respect to device j can be related to θ_i, the orientation of device i with respect to the the coordinate system used by the local host:

$$\psi_{ij} = \phi_{ij}(x_i, y_i, x_j, y_j) - \theta_i, \quad (2)$$

where ϕ_{ij} is the angle of the vector drawn from device i to device j, as shown in figure 7. As indicated in the equation, ϕ_{ij} is a function of the locations of the devices i and j, and can be calculated using trigonometry.

Since the devices in the system report measurements of the ranges d_{ij} and the angles-of-arrival ψ_{ij}, a *non-linear regression* algorithm can be applied to arrive at estimates for the devices' 2D relative locations and orientations. The regression process converges toward location and orientation estimates which minimise the sum of the squares of the *residuals* (i.e. the difference between devices' measured ranges and angles, and the ranges and

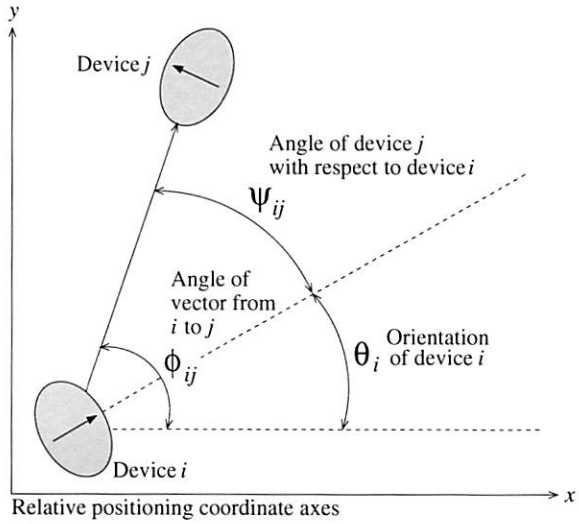


Figure 7: Device angle relationships.

angles as calculated using equations 1 and 2). The solution can be further refined by using *Studentized residuals* to identify ranges and angles which are likely to have large errors, and repeating the regression with those measurements discarded. The algorithm also creates an estimate of the *standard error* of the solution set, based on the residuals of the final data used. This error estimate is essentially a measure of "goodness of fit," and can be used to discard location and orientation solution sets which do not statistically fit the data supplied. These techniques have previously been shown to work well in positioning systems [27].

The positioning stage operates on all sensor information available in the model and produces new position and orientation data for all devices represented in the model at that point in time. Note that this data is described in relative coordinates referenced to the local host on which the model is computed.

4.2.3 Abstraction

The third stage of the pipeline performs various abstractions on the quantitative output of the previous stage in order to generate qualitative information. The relative coordinates are transformed into qualitative relations that encode spatial relationships, which are closer to human concepts of space [6]. The abstraction stage computes both angular and distal relations. The former operate on the orientation information, and compute relations such as `left_of` based on angular deviation from a cardinal direction. The latter are computed from the coordinates using threshold values to partition distances into categories such as `nearby` or `far_from`. The output of this stage hence consists of a set of static spatial relations, which are inserted in the model before the final stage of the pipeline is triggered.

4.2.4 Inference

The inference stage deals with change over time. It takes the output of the abstraction stage and compares the set of relations to those contained in previous graphs in the list. The inference process is based on first order logic and generates a set of spatiotemporal relations such as `approaching` or `moving_away` [13]. It also identifies events such as the arrival of new nodes in the Relate network or the departure of nodes. For example, a node is assumed to have moved away from the Relate network when no sensor data for the node has been registered over a certain period of time. The computation of these relations constitutes the final step in the Relate graph generation cycle. After completion of this stage, the current graph is timestamped. It remains stored in the model but gets moved further back in the list as newer graphs are added.

4.3 Host-specific configuration

Most of the information maintained in the Relate model is derived from processing of sensor observations in the pipeline described above. Some attributes however need to be configured when a Relate host is first set up. These include host type and dongle orientation with respect to the host device. Relate users can set these attributes through a graphical configuration tool. The tool allows users to describe where the Relate dongle is physically connected on their mobile device. This information is required for correct spatial mapping of sensor observations, but cannot be known in general as different devices have their USB connectors in different places.

5 Service Layer and Application

The service layer provides spatially-aware applications with access to the Relate model. Three services are available. The **event service** implements an asynchronous communication channel between the model and the application. It allows applications to subscribe to model updates and to receive associated event notifications. The **query service** provides applications with read-only access to the model. Applications can read specific attribute values associated with nodes and relations or they can get a copy of the entire model. The **spatial communication service** implements a spatially-aware group communication mechanism for Relate applications.

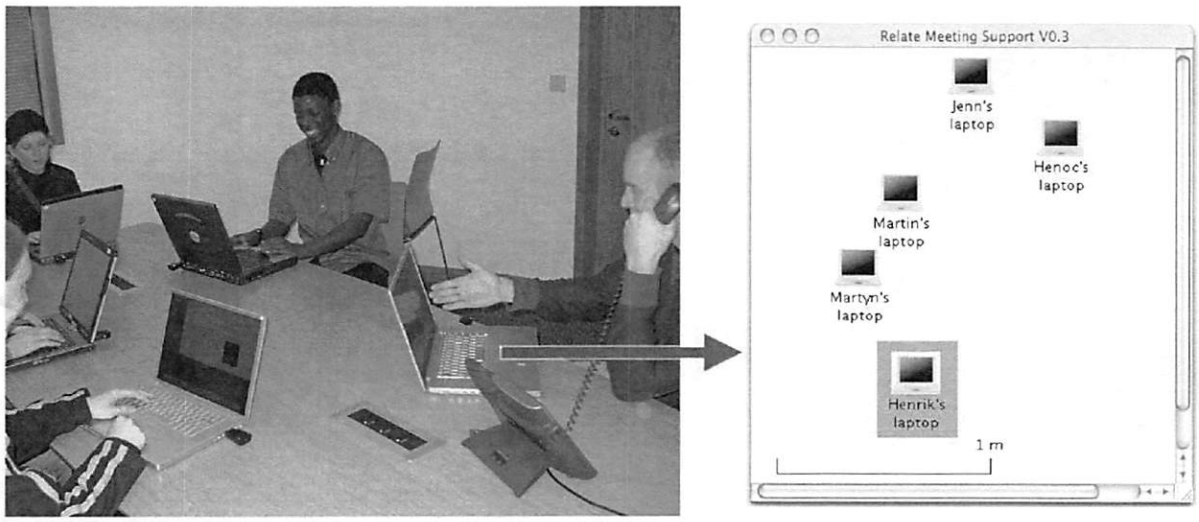


Figure 8: The awareness tool provides participants in a meeting with a visualisation of the spatial arrangement of devices and their users.

5.1 Event Service

To stay informed about the dynamic status of a Relate network, Relate applications can subscribe to three different event types. **Model events** are generated whenever the spatial model has been updated. Applications can subscribe to events informing them of updates of the value of specific attributes of individual nodes or relations. Similarly, applications can also subscribe to receive a copy of the entire model whenever a change has occurred. **Spatial events** are generated whenever the spatial configuration of Relate devices changes in some significant way. For example, applications can register to receive notifications whenever the distance to another device exceeds or falls below a specified threshold. Finally, **network events** are generated to inform applications about the condition of the Relate network. In particular, applications can register to be notified when a device joins or leaves the network.

A spatial event subscription contains two arguments: a predicate and a notification mode. Predicates are expressions of spatial situations such as `distance(device1, device2) < 1 m`. They define in which situation a notification should be sent. The notification mode determines how often a notification needs to be sent. There are two modes: `once` indicates that a notification should be sent only the first time a predicate holds; `continuous` mode indicates a notification should be sent as long as the predicate holds. For example, to be notified when a device comes within one metre of the local device, an application would specify the predicate `distance(local,` x`) < 1` and notification mode `once`. (By definition, `local` is a built-in constant referring to the local device, and x is a variable which refers to any device matching the criterion.) To be notified repeatedly as long as there is a device in front of the local device, the application would specify the condition `in_front_of(local,`x`)` with mode `continuous`. The actual notification frequency depends on the update rate of the model, which in turn is determined by the availability of new sensor data.

5.2 Query Service

The query service provides applications with read-only access to the model. Applications can read attribute values of individual nodes or they can get a copy of the entire model. Examples of some query methods are as follows: `getDeviceList()` returns list of all available devices; `getDeviceCoordinates()` returns coordinates of a device; and `getDevicesInFront()` returns devices in front of the current device.

These queries retrieve information from graphs stored in the model. There is a time parameter in each call which specifies which data should be retrieved: the most recent graph can be accessed by specifying the constant `now`, a positive number indicates an absolute timestamp (for example "10h30m17s") and a negative number indicates a relative time period (i.e. "-30s" refers to data that is thirty seconds old). If there is no exact match between the specified time and the timestamps of the graphs, the graph whose timestamp is closest is selected.

5.3 Spatial Communication Service

The communication service realises an asynchronous communication mechanism for Relate applications. It provides a number of spatially-aware communication primitives that can be used to disseminate data between Relate devices. For example, `send(host_name, msg)` sends a message to a single named device; `sendFront(msg)` sends a message to all devices currently in front of the local device; and `sendBeyondDistance(float, msg)` sends a message to all devices that are currently more than a specified distance away from the local device.

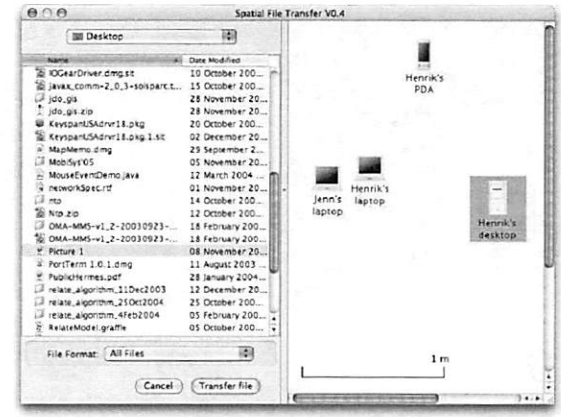


Figure 9: Spatial file transfer

5.4 Application Prototyping

An "awareness tool" can be used to provide users with a visualisation of the arrangement of nearby Relate devices (figure 8). For example, the name of each device's owner or user can be displayed underneath the device. Using this view, people can easily identify each other in a meeting scenario where participants sit around a table with their computers in front of them. This is relevant in situations where participants may not know one another by name.

The awareness tool makes use of the query and event services. At startup it requests a copy of the complete model and builds the initial visualisation using the `getDeviceList(now, ...)` and `getDeviceCoordinates(now, ...)` query primitives. Then, it uses the event service to stay informed about the coordinates of each device and about newly appearing or disappearing devices. To subscribe to the coordinates of all devices the tool uses the following subscription: predicate `coordinate(`x`)` with mode `continuous`. To receive a notification when a device enters or leaves the Relate network, the tool can use the following two subscriptions: `appearing(`x`)` and `disappearing(`x`)` (in both cases with mode `once`).

Note that the awareness tool does not require IP connectivity between devices. All information displayed is communicated over the dongle network, including the user name.

Building on the view provided by the awareness tool, we have prototyped a Relate file transfer application. It allows users to copy files between computers having IP connectivity. The user interface is divided into two panels (figure 9). The left panel is a file browser which is used to select a file on the local computer. The right panel shows the awareness tool view, i.e. the spatial arrangement of nearby Relate devices. Here the user selects the device where the file should be transferred to. The file transfer is initiated by clicking the 'Transfer File' button. Files are copied to a fixed destination folder on the target machine.

The visualisation of the spatial arrangement simplifies the task of selecting a target computer. In particular, the user does not need to know the IP address or name of the computer to which he or she wants to connect. Instead, the graphical visualisation of the spatial arrangement of devices enables the user to compare what they see in the real world (the computers on the table) with what they see on the display. Using the spatial arrangement as the selection criteria is more intuitive than traditional methods of selecting a computer (manually typing an IP address or selecting a computer name from a list) and it works in cases where the name or address of the target computer is not known by the user.

This application makes use of all three services. The visualisation is generated using the query and event services (as described above); the file transfer is performed using the spatial communication service.

6 System Evaluation and Discussion

Tests were performed to characterise the performance of the Relate system. Using five laptops equipped with Relate dongles, measurements were taken on a 2.4×1.6 m surface in an indoor office environment, shown in figure 10. The laptops used were as follows: two Dell Inspirons and an Acer tablet PC running Windows XP, one Dell Inspiron running Linux, and an Apple iBook running Mac OS X.

For every experiment, each laptop was placed at a randomly generated location and orientation. The actual locations and orientations of the dongles were measured manually with the aid of a reference grid on the surface. The system was then allowed to run for about seven minutes, with positioning model evaluations being triggered every half second on each computer. A software

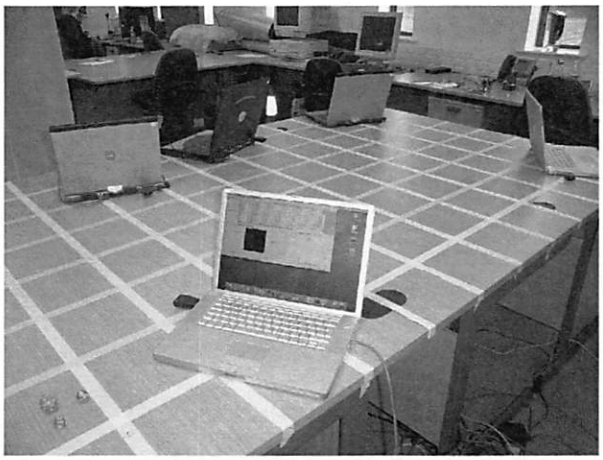


Figure 10: The system was evaluated in experiments using five laptops equipped with Relate dongles.

tool which accesses the raw measurements and models through the Relate service layer was used to log the data during the course of each experiment.

In all, over one hundred such experiments were performed. In half of the experiments, the randomly-generated locations and orientations were selected using the constraint that the dongles all have line-of-sight to one another. USB ports on mobile computers are often placed on outward-facing edges, and the dongles protrude by about five centimetres, making line-of-sight achievable in many situations. Nonetheless, it is important to also characterise poor line-of-sight conditions. Thus, the other half of the experiments were conducted with limited line-of-sight between the dongles. For each of these experiments, three out of the ten possible lines of sight between the five dongles were blocked due to the orientation of the laptops.

6.1 Sensor layer

This subsection presents results characterising aspects of the Relate sensor layer.

6.1.1 Raw measurement error

Figure 11 shows the error distribution of the raw range and angle-of-arrival measurements reported by the Relate dongles to their hosts. In good line-of-sight (LOS) conditions, the raw measurements are accurate to within 9 cm and 33° in 90% of cases. As with any ultrasonic ranging device, limited line-of-sight conditions cause a degradation in performance; error in these cases is about 11 cm and 48° with 90% confidence. When line-of-sight between two devices is fully or partially blocked, several factors can contribute to measurement error. First, the tendency of acoustic waves to bend around obstructions can lengthen the measured time-of-flight, reduce the received signal strength, and cause the received pulse shape to vary from the expected shape of a direct-path pulse. Second, the receiver is more likely to identify multipath signals (i.e. reflections) as the valid ranging pulse.

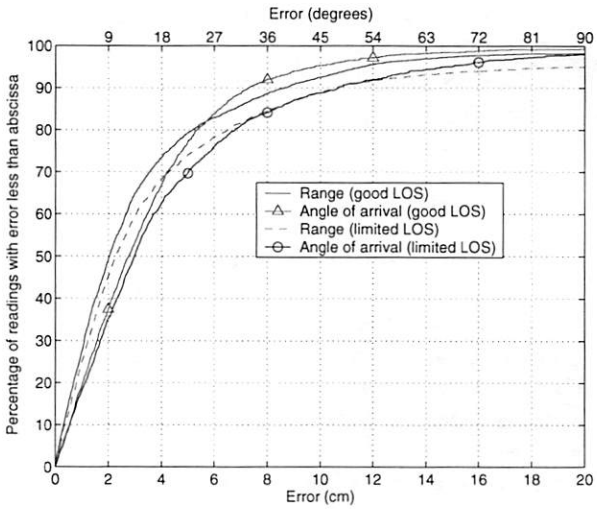


Figure 11: Raw measurement error distributions.

6.1.2 Successful transmission rate

As described in section 3.2, seven 13 ms radio slots are used by each dongle during its transmission sequence. Since the dongles share their views of the network state, each dongle knows how many other dongles are in the system, and waits the required time for the others to transmit before attempting to transmit again. With perfect RF connectivity between the five dongles used in the experiments, a delay of about 450 ms between successive transmissions for a given dongle would be expected.

Figure 12 shows the distribution of the actual times between ultrasonic transmissions. Although the median of the distribution is almost exactly the expected 450 ms, there is some variation due to imperfect RF communication between the dongles. The range of the RFM transceivers used in the dongles is nominally thirty metres indoors. However, this relies upon the use of a certain length antenna, and no nearby obstructions which may cause RF attenuation or multipath. Our implementation utilises a relatively short antenna to minimise obtrusiveness, and the dongles are placed on a surface which tends to further degrade radio communication.

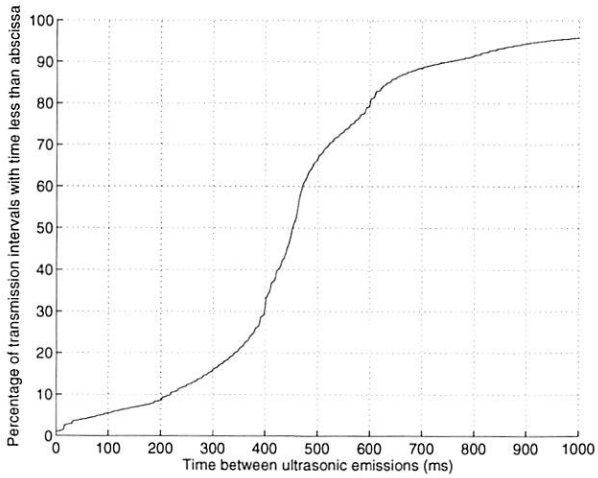


Figure 12: Distribution of the times between the ultrasonic transmissions of all dongles during all experiments.

6.1.3 Start-up delay

Twenty "start-up" experiments were performed to explore how long it takes for newly activated dongles to join the Relate network. In these tests, one host device was chosen at random to shut down its dongle. The dongle was then subsequently re-started by the host, in order to simulate a dongle that has just been plugged in. In 95% of cases, the other four dongles in the system successfully detected a radio trigger packet from the newly-joined dongle within 3.7 s after it was started up. The start-up sequence need only be executed once per session, and the delay is well within reasonable expectations for peripheral start-up times—typical USB devices take several seconds to be recognised by the operating system.

6.2 Model layer

In this subsection, the performance of the model layer is characterised. The abstractions and qualitative spatial relationships computed in the model rely upon the relative location and orientation results of the non-linear regression algorithm. Thus, the bulk of the analysis presented here focuses on the regression results.

6.2.1 Location and orientation estimation

As mentioned in section 4.2.2, the regression algorithm uses Studentized residual analysis to aid in eliminating range and angle-of-arrival measurements which do not fit well with the other measurements. This means that the relative location and orientation results returned by the algorithm have the potential to be more accurate than the raw range and angle-of-arrival measurements reported by the dongles, as shown in figures 13 and 14.

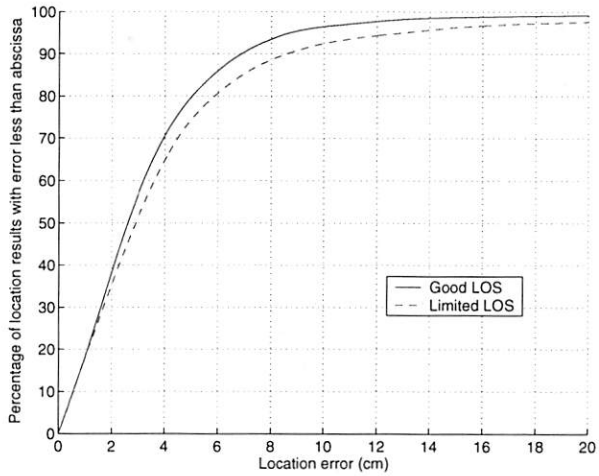


Figure 13: Relative location error distributions.

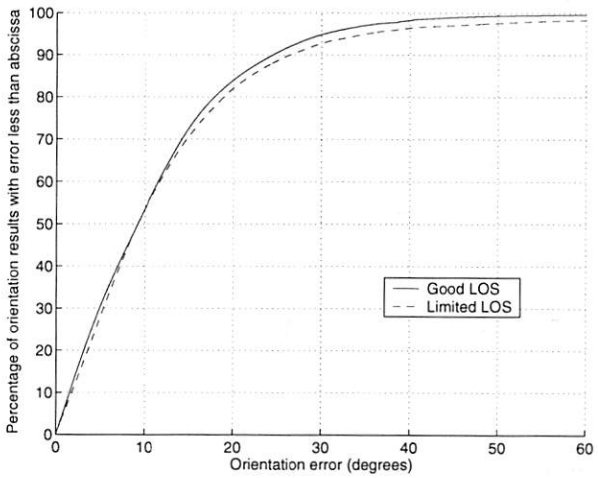


Figure 14: Relative orientation error distributions.

The plots for "good LOS" and "limited LOS" show the regression algorithm's error when operating on the data reported by each host's dongle independently for the two line-of-sight cases. As described in section 3.4, each dongle relays over USB (1) measurements it has taken using its transducers and (2) measurements taken by other dongles which they have broadcast over the dongle network. The performance of the two line-of-sight cases is summarised in table 2. As expected, the best performance is to be had when good line-of-sight conditions exist. In such cases, the estimates are better than 7 cm and 25° for 90% of the results returned.

Sets of location and orientation estimates which had a high *standard error* (section 4.2.2) were rejected by

the algorithm, and no location/orientation updates were made in the model layer. Limited line-of-sight conditions cause the algorithm to return location and orientation estimates with acceptable standard error about 70% of the time, as shown in the last column of table 2. Location and orientation estimates made during good line-of-sight conditions were more successful, with the regression algorithm returning a result about 88% of the time.

6.2.2 Moving devices

Accurate, up-to-date location and orientation information is particularly important when devices are on the move. As part of each experiment, a randomly chosen laptop was moved from one location to another on the surface. The approximate time at which the laptop came to a stop was logged, in addition to the dongle's new location and orientation on the surface. In about 15% of the motion experiments, the relative positioning solutions were rejected because they had high standard error estimates. However, in the motion experiments where a valid solution was returned, 70% returned an accurate set of readings within one second after the laptop had come to rest, 90% within five seconds, and 95% within six seconds.

6.3 Discussion

This subsection highlights several issues for further discussion.

6.3.1 Signal processing limitations

The accuracy of the range measurements reported by the dongles is limited by the sampling and computational constraints of the PIC microcontroller. A production version of the system might use faster sampling hardware and a microcontroller more suited to signal processing; this would allow pulse coding and matched filtering techniques to be used to achieve typical range accuracies of 3 cm or better.

6.3.2 Improving the fraction of readings returned

Limited line-of-sight conditions affect the rate that new location and orientation updates are applied to the model, since the regression algorithm rejects far more of its computed solutions due to a high standard error estimate. Often, a set of successive regression evaluations are rejected because a number of measurements held in the model are erroneous, and it takes some time for either (1) the measurement to be replaced by a newer, more accurate one, or (2) for the measurement to become old enough to be deleted from the model. Currently, the regression algorithm gives no direct feedback to the model about the measurements. Since the regression algorithm uses the residuals of the measurements to gauge their correctness, one appropriate strategy would be for the regression algorithm to flag measurements having large Studentized residuals. If a particular measurement is repeatedly flagged during several consecutive regression evaluations, it could be struck from the model altogether in order to decrease the number of solutions rejected by the algorithm during subsequent evaluations.

6.3.3 Scalability

The dongle protocol is designed to support systems of up to twenty dongles, and we have experimentally verified with ten dongles that the system continues to function as expected. However, because the system operation utilises a time-division scheme, its responsiveness begins to suffer when there are larger numbers of dongles present. For example, in a system with thirty dongles, each would only transmit about every four seconds. This can produce quite long delays for up-to-date location estimates to be returned, especially if line-of-sight between the dongles is limited.

6.3.4 Kalman filtering

As described previously, our implementation of the model layer gathers a number of measurements and every so often evaluates location and orientation using non-linear regression. Although many sets of measurements stored in the model cover a short span of time (typically less than a second), there is still a non-negligible latency between each successive measurement. A constant feed of measurements such as this (often described as "single-constraint-at-a-time") is a classic application for Kalman or particle filters. Such methods may be able to provide better continuity between successive location and orientation estimates, since locations and orientations would be updated appropriately with each incoming measurement, and the time difference between measurements would be properly taken into account.

7 Related Work

The Relate system described in this paper is distinct in its capability to provide fine-grained, peer-to-peer relative positioning for mobile devices. However, the work is closely related to a number of research efforts which are concerned with sensing and modelling to support spatially-aware behaviour.

An overview of location systems and technologies for mobile and ubiquitous computing is given in [8]. Many of the available location technologies and systems provide information at metre– or room-level accuracy which

Table 2: Model layer accuracy summary.

	Ninetieth percentile		Ninety-fifth percentile		Fraction of
	Location	Orientation	Location	Orientation	readings returned
Good LOS	6.9 cm	25°	8.7 cm	30°	87.9%
Limited LOS	8.6 cm	26°	13.0 cm	35°	70.7%

has been shown to be useful for a wide range of mobile tasks, including discovery of device and user co-location within a certain space or area [14, 22]. However, only a few systems reported to date are capable of providing more fine-grained spatial information to devices and users that are already co-located, as targeted by our system. This includes systems using computer vision [3, 15], ultra-wide band radio [4] and ultrasonic ranging [18, 19, 27]. As discussed in the introduction, these systems have the disadvantage of being reliant upon infrastructure deployed in the environment.

DOLPHIN is a location system that utilises peer-to-peer positioning of sensor nodes to provide a more flexible sensing infrastructure in comparison to other indoor location systems [16]. The DOLPHIN devices are particularly similar to the Relate dongles in that they perform bidirectional ultrasonic ranging. In terms of hardware, the DOLPHIN design focuses on omnidirectional range measurements whereas the Relate dongles are optimised for approximately co-planar operation. Relate dongles can also measure pulse angle-of-arrival which helps to more tightly constrain the regression solution, as well as provide orientation information.

In terms of overall system operation, DOLPHIN has two attributes: (1) each node computes only its own location using its own ranging measurements, and (2) the location results produced are absolute, based on a minimal number of reference nodes placed in the environment. In contrast, a Relate dongle collects measurements reported by other dongles (in addition to its own) and passes these to its host, which is responsible for computing purely relative positioning results for *all* devices in the system. As a result of their contrasting methods of operation, the ad hoc protocols employed by the two systems are fundamentally different.

Close in spirit to our work are a number of approaches that are focused on relative positioning as opposed to absolute location. Approaches to modelling proximity in mobile computing utilise Bluetooth and IrDA device discovery, WiFi cell ID, or radio signal strength. Systems range from bespoke awareness devices that alert proximity of "friends" [7, 11] to more general frameworks such as the NearMe wireless proximity server [14]. NearMe uses WiFi signals to model device and user proximity by comparing their lists of detected base stations and their signal strengths. An advantage of NearMe is that it does not require additional dedicated sensors but the accomplished accuracy is metre-scale.

The Relate system design emphasises provision of a complete framework from the sensor layer through the model layer to application services. In this respect, it shares an overall approach with the Sentient Computing project [1], the Location Stack [9], and NearMe [14].

8 Conclusion

The Relate system extends mobile computing devices with a distinct new capability, enabling them to *directly* establish their spatial relationships when they become co-located. Mobile computers that are augmented with a Relate dongle and software system can acquire fine-grained information about their position and orientation relative to other devices nearby, without need for any infrastructure installed in the environment. This is achieved by the packaging of ultrasonic sensing technology in a novel way as a USB dongle ready for use with everyday mobile computers.

A key feature of the system is the vertical integration, going from the sensor network layer through spatial modelling to provision of a set of application services. This encompasses a variety of methods arranged in a data processing pipeline to extract quantitative as well as qualitative spatial information at various levels of abstraction. The targeted application settings are multi-device and multi-user interaction, and we have included examples in this paper to illustrate how applications can be built on top of the Relate system.

Our experimental results have shown that the sensor and model layers provide relative location and orientation estimates at an accuracy and update rate appropriate for the scenarios we envision; the 90% accuracy is about 8 cm and 25°, and up-to-date estimates can be produced several seconds after a device has been moved.

Acknowledgements. The authors would like to thank Martyn Welch for developing the algorithm which estimates ranging signal angle-of-arrival at the dongle. The work presented in this paper is part of a project (GR/S77097/01) funded by the UK Engineering and Physical Sciences Research Council.

References

[1] ADDLESEE, M., CURWEN, R., HODGES, S., NEWMAN, J., STEGGLES, P., WARD, A., AND HOPPER, A. Implementing a sentient computing system. *IEEE Computer 34*, 8 (Aug. 2001), 50–56.

[2] BARTON, J., JOHANSON, B., VIJAYARAGHAVAN, V., HSIEH, T., SHIMIZU, T., AND FOX, A. The MeetingMachine: Interactive workspace support for nomadic users. In *Proceedings of the Workshop on Mobile Computing Systems and Applications* (Monterey, USA, Oct. 2003), pp. 2–12.

[3] BRUMMIT, B., MEYERS, B., KRUMM, J., KERN, A., AND SHAFER, S. EasyLiving: Technologies for intelligent environments. In *Symposium on Handheld and Ubiquitous Computing* (Bristol, UK, 2000), Springer, pp. 25–27.

[4] CADMAN, J. Deploying commercial location-aware systems. In *Proceedings of the 2003 Workshop on Location-Aware Computing (held as part of UbiComp 2003)* (Seattle, USA, Oct. 2003), pp. 4–6.

[5] CAMARATA, K., DO, E. Y.-L., JOHNSON, B. D., AND GROSS, M. D. Navigational blocks: Navigating information space with tangible media. In *Proceedings of the International Conference on Intelligent User Interfaces* (San Francisco, USA, Jan. 2002), pp. 31–38.

[6] CLEMENTINI, E., FELICE, P. D., AND HERNÁNDEZ, D. Qualitative representation of positional information. *Artificial Intelligence 95*, 2 (1997), 317–356.

[7] DAHLBERG, P., LJUNGBERG, F., AND SANNEBLAD, J. Supporting opportunistic communication in mobile settings. In *CHI 2000 Extended Abstracts on Human Factors in Computing Systems* (The Hague, The Netherlands, 2000).

[8] HIGHTOWER, J., AND BORRIELLO, G. Location systems for ubiquitous computing. *IEEE Computer 34*, 8 (Aug. 2001), 57–66.

[9] HIGHTOWER, J., BRUMITT, B., AND BORRIELLO, G. The Location Stack: A Layered Model for Location in Ubiquitous Computing. In *Proceedings of the 4th IEEE Workshop on Mobile Computing Systems and Applications* (Callicoon, NY, USA, 2002), pp. 22–28.

[10] HOFFMANN, F., AND SCOTT, J. Location of mobile devices using Networked Surfaces. In *Proceedings of UbiComp: Ubiquitous Computing* (Göteborg, Sweden, Sept. 2002), pp. 281–298.

[11] HOLMQUIST, L., FALK, J., AND WIGSTRÖM, J. Supporting group collaboration with inter-personal awareness devices. *Journal of Personal Technologies 3*, 1–2 (1999).

[12] JACOB, R., ISHII, H., PANGARO, G., AND PATTEN, J. A tangible interface for organizing information using a grid. In *Proceedings of the Conference on Human Factors in Computing Systems* (Minneapolis, USA, Apr. 2002), pp. 339–346.

[13] KRAY, C., AND BLOCHER, A. Modeling the basic meanings of path relations. In *Proceedings of the 16th IJCAI* (San Francisco, CA, USA, 1999), Morgan Kaufmann, pp. 384–389.

[14] KRUMM, J., AND HINCKLEY, K. The NearMe wireless proximity server. In *Proceedings of Ubicomp: Ubiquitous Computing* (Nottingham, UK, Sept. 2004), Springer, pp. 283–300.

[15] LÓPEZ DE IPIÑA, D., MENDONÇA, P., AND HOPPER, A. TRIP: A low-cost vision-based location system for ubiquitous computing. *Personal and Ubiquitous Computing 6*, 3 (May 2002), 206–219.

[16] MINAMI, M., FUKUJU, Y., HIRASAWA, K., YOKOYAMA, S., MIZUMACHI, M., MORIKAWA, H., AND AOYAMA, T. DOLPHIN: a practical approach for implementing a fully distributed indoor ultrasonic positioning system. In *Proceedings of Ubicomp: Ubiquitous Computing* (Nottingham, UK, Sept. 2004), Springer, pp. 347–365.

[17] PATTEN, J., ISHII, H., HINES, J., AND PANGARO, G. Sensetable: a wireless object tracking platform for tangible user interfaces. In *Proceedings of the Conference on Human Factors in Computing Systems* (Seattle, USA, Apr. 2001), pp. 253–260.

[18] PRIYANTHA, N. B., MIU, A. K. L., BALAKRISHNAN, H., AND TELLER, S. The Cricket Compass for context-aware mobile applications. In *Proceedings of the Seventh International Conference on Mobile Computing and Networking* (Rome, Italy, July 2001).

[19] RANDELL, C., AND MULLER, H. L. Low cost indoor positioning system. In *Proceedings of the 3rd international conference on Ubiquitous Computing* (Atlanta, GA, USA, 2001), Springer, pp. 42–48.

[20] REKIMOTO, J., AND SAITOH, M. Augmented surfaces: A spatially continuous work space for hybrid computing environments. In *Proceedings of the Conference on Human Factors in Computing Systems* (Pittsburgh, USA, May 1999), pp. 378–385.

[21] ROSENFELD, D., ZAWADZKI, M., SUDOL, J., AND PERLIN, K. Physical objects as bidirectional user interface elements. *IEEE Computer Graphics 24*, 1 (2004), 44–49.

[22] SCHILIT, B. N., ADAMS, N. I., AND WANT, R. Context-aware computing applications. In *Proceedings of the Workshop on Mobile Computing Systems and Applications* (Santa Cruz, CA, USA, December 1994), IEEE Computer Society, pp. 85–90.

[23] SCHMIDT, A., STROHBACH, M., VAN LAERHOVEN, K., FRIDAY, A., AND GELLERSEN, H.-W. Context acquisition based on load sensing. In *Proceedings of UbiComp: Ubiquitous Computing* (Göteborg, Sweden, Sept. 2002), pp. 333–350.

[24] STREITZ, N., GEISSLER, J., HOLMER, T., KONOMI, S., MÜLLER-TOMFELDE, C., REISCHL, W., REXROTH, P., SEITZ, P., AND STEINMETZ, R. i-LAND: An interactive landscape for creativity and innovation. In *Proceedings of the Conference on Human Factors in Computing Systems* (Pittsburgh, USA, May 1999), pp. 120–127.

[25] ULLMER, B., AND ISHII, H. The MetaDESK: Models and prototypes for tangible user interfaces. In *Proceedings of the Symposium on User Interface Software and Technology* (Banff, Alberta, Canada, Oct. 1997), pp. 223–232.

[26] UNDERKOFFLER, J., AND ISHII, H. Urp: A luminous-tangible workbench for urban planning and design. In *Proceedings of the Conference on Human Factors in Computing Systems* (Pittsburgh, USA, May 1999), pp. 386–393.

[27] WARD, A., JONES, A., AND HOPPER, A. A new location technique for the active office. *IEEE Personal Communications 4*, 5 (Oct. 1997), 42–47.

Notes

[1] More detail on Smart-Its Particles is available at `http://particle.teco.edu/`.

[2] The particular dongles we describe in this paper assume that the USB port on the host device is oriented horizontally, but implementations for vertical USB ports are equally feasible.

[3] The size of the network state table and the number of measurements broadcast increase with the number of dongles in the system. To be able to transmit this information fully, dongles dynamically change the number of RF slots used depending on the number of dongles present. For example, two slots are utilised by systems with up to ten dongles, whereas systems with eleven to nineteen dongles utilise three slots.

WALRUS:
Wireless Acoustic Location with Room-Level Resolution using Ultrasound

Gaetano Borriello[1,2], Alan Liu[1], Tony Offer[1], Christopher Palistrant[1], Richard Sharp[3]
[1]*Department of Computer Science & Engineering, University of Washington, Seattle, WA [USA]*
[2]*Intel Research Seattle, Seattle, WA [USA]*
[3]*Intel Research Cambridge, Cambridge [UK]*
{gaetano@cs.washington.edu}

Abstract

In this paper, we propose a system that uses the wireless networking and microphone interfaces of mobile devices to determine location to room-level accuracy. The wireless network provides a synchronizing pulse along with information about the room. This is accompanied by an ultrasound beacon that allows us to resolve locations to the confines of a physical room (since audio is mostly bounded by walls). We generate the wireless data and ultrasound pulses from the existing PCs in each room; a PDA carried by a user listens for both signals. Thus, our approach does not require special hardware. We do not use ultrasound to send data. As a result we dramatically reduce the computational burden on the mobile device while also decreasing the latency of location resolution. Our results indicate that (i) ultrasound detection is robust even in noisy environments with many reflective surfaces; and (ii) that we can determine the correct room within a couple of seconds with high probability even when the ultrasound emitting PCs are not synchronized.

1. INTRODUCTION

Future mobile devices will need the ability to determine their location and, thus, enable location-enhanced computing. Location is a major part of a user's context and applications can be constructed that adapt to the user's current location. For example, a calendar reminder system can adapt by adjusting the time of an alarm based on traffic conditions or public transportation options between the user's current location and their next destination. Applications can be designed that record the current location so as to better classify data for future retrieval. For example, a digital camera can record the location at which each picture was taken. Location can also be used to modify the behavior of existing applications. For example, a web browser can be set up to automatically render web pages associated with the user's current location.

The Global Positioning System (GPS) is by far the most prevalent example of a location system. It uses signals from synchronized orbiting satellites to calculate a three-dimensional position relative to Earth's coordinate system. There are two issues with GPS that limit its utility in ubiquitous and mobile computing scenarios. First, it requires line-of-sight to at least three satellites for 2-D location resolution (four for 3-D). Unfortunately, it is difficult to obtain line-of-sight in most environments where users spend most of their time (i.e., indoors) and in places where most users live and work (i.e., urban centers). Second, and more importantly, a 3-D coordinate does not help a user locate what they need as that coordinate must be translated to a form that is understandable to a person. For example, knowing that someone is 100 meters above sea level at 47°N and 122°W is much less useful than knowing they are in room 572 of the Allen Center on the campus of the University of Washington. Clearly, the information in the latter is much more useful in finding people and services.

Many systems have been designed to provide mobile devices with the capability to monitor their location indoors (some of these will be discussed in detail in the next section and a more complete bibliography can be found at: *http://binary.engin.brown.edu/publication/Positioning_Ref.pdf*). Designers of these location systems need to make several key tradeoffs that affect the system's usability [5, 7], among these are:

- **Affordability.** A location system should be a minute fraction of the total cost of a mobile device. Cost includes not only the final monetary cost to individual users, but also the cost associated with installation, management, and maintenance of the infrastructure portion of the system.
- **Resource Requirements.** Mobile devices have limited memory, computational capabilities, and

power; the need to accommodate expensive computations not only adds extra cost to the system but also makes it less usable if it shortens battery life.

- **Privacy.** A system that requires a user or mobile device to query a server or host for a location will need to reveal the user's identity in exchange for this information. This may be considered an undesirable feature for users who wish to remain anonymous. Moreover, the infrastructure-based supplier of this information may charge the user for this service thereby limiting the number of applications and/or their frequency of location updates. With careful design, a system can be devised where a user receives information that helps determine their location without potentially revealing confidential information or even their presence.
- **Portability.** Mobile systems are an evolving technology and some consideration should be given to ensure that a system can be easily maintained during upgrades and across most platforms. To ensure that a system is readily adopted and maintained across several generations of hardware, a location system should consider how and if the system will be able to adapt to future technologies.
- **Precision.** Designers must decide what degree of precision a location system will provide. Precision is defined as the granularity that a system is capable of measuring. Many location systems have been developed with precisions ranging from centimeters to kilometers. For many ubiquitous computing applications, room-level accuracy is an important grain size as it closely relates to the places people often think about. Usually, higher levels of precision correlate strongly with increased cost of the location system.
- **Accuracy.** Designers of location aware systems consider accuracy to be the percentage of the time a known level of precision is reached. For example, a GPS receiver that has a precision level of 15 meters might be accurate 95% of the time in an open field; however, system accuracy will diminish within an office building. Accuracy at the room-level is very important as, for example, it is not acceptable for a context-aware location system to inadvertently connect a user's laptop to the data projector that is in the conference room next door.

Based on these design factors, many different types of systems can be developed that will meet a variety of unique criteria in the available design space. This paper describes, WALRUS, a location system that emphasizes: low cost, high privacy, high portability, room-level precision, and high accuracy.

A key design feature of WALRUS is that it leaverages existing hardware. The WALRUS client can run on any device that can receive WiFi packets and listen to ultrasound (at approximately 21KHz). These capabilities are found in most modern laptops, tablets, and PDAs that usually include integrated WiFi and microphones/speakers. Furthermore, they are likely to make their way into even more devices, such as cell phones and wrist-watches, in the near future with the advent of low-power radio protocols such as 802.15.4 (Zigbee) and ultra-wide-band (UWB).

Section 2 of this paper discusses several other location systems that share similarities with the WALRUS system, but as will be seen later, exhibit important differences as well. Section 3 describes the implementation of WALRUS. Section 4 details the results of our experiments and evaluates how well the system worked. Finally, Section 5 outlines future work that can be done to improve the WALRUS systems and how it may evolve.

2. RELATED WORK

The problem of determining a device's location has been the topic of countless research endeavors, all of which have had to balance the various tradeoffs between affordability, privacy, portability, precision, and accuracy. As in all engineering disciplines, tradeoffs have to be made; in order to improve one aspect of a project, another aspect must be compromised to some degree. There are no location-sensing technologies that excel at everything. This section describes the strengths and weaknesses of several existing location-sensing technologies in terms of the following attributes: cost, privacy, precision, and operational scope. We will compare the WALRUS system to each of the technologies described in Table 1.

GPS uses time-of-flight calculations from orbiting satellites to triangulate the position of mobile receivers near the surface of the Earth. GPS is similar to WALRUS in that there is no centralized system that tracks the location of the mobile devices. However, GPS operates at a much larger and much more expensive scale than WALRUS. It costs billions of dollars to establish the infrastructure for GPS and the mobile receivers usually cost on the order of USD100.GPS can determine location with a precision of 1 to 5 meters [5]. User privacy is respected since

System	*Privacy*	*Client cost*	*Infrastructure cost*	*Precision*	*Operational Scope*
WALRUS	Client devices compute location	No additional hardware	PC per room + WiFi	room-level	Indoor
GPS	Client devices compute location	Approx. USD100 receiver	Satellite network	1-5m	Outdoor
Place Lab	Client devices compute location	No additional hardware	WiFi, GSM, and/or Bluetooth beacons	15-30m	Outdoor/Indoor
Radar	Client devices compute location	No additional hardware	WiFi coverage	5m	Indoor
ActiveBadge	Central server tracks clients	Low-cost badge	Badge IR receivers	room-level	Indoor
CoolTown	Client devices talk to appliances	IR transceiver	IR transmitter on all objects of interest	3-5m IR	Indoor
Active Bat	Central server tracks clients	Low-cost bat	Ultrasound receivers in ceiling + RF link	5-10cm	Indoor
Cricket	Client devices compute location	Low-cost client ultrasound rcvr	Ultrasound transmitters	1 m^2 region	Indoor
E911/E112	Central server tracks clients	No additional hardware	Cellular network	100m	Outdoor/Indoor

Table 1. Summary of location sensing systems.

GPS position information is computed on the mobile devices and the user is in full control of whether their location is reported to others. The infrastructure has no knowledge of who is receiving the signal. One of the weaknesses of GPS is that it does not function indoors or in urban canyons (spaces between tall buildings) without modification since line-of-sight is required for the satellite communications to the mobile receivers.

These modifications include the installation of satellite repeaters (or *pseudo-lites*) to serve a particular area such as a building. Recent advances in GPS receiver technologies are making it possible to receive the signals indoors as well but only at a greatly reduced resolution (due to multi-path effects).

Like GPS, Place Lab provides absolute coordinates describing the position of a device [9]. It functions by determining nearby radio sources (such as 802.11 access points, GSM cell towers, fixed Bluetooth devices, etc.) and looking up their MAC addresses in a client-side database. A position is estimated based on the pattern of beacons seen over time. Place Lab shares with WALRUS the goal of using existing hardware and does not require the purchase of any new devices but, unlike WALRUS, Place Lab provides absolute coordinates rather than room determinations. It is likely that, in an indoor situation, the uncertainty radius of an absolute coordinate will cross several room boundaries as Place Lab's precision is on the order of 15-30m, making it unclear whether the device is in one room or another. Place Lab is privacy-observing in that location determination is done on the client. There is no centralized infrastructure that tracks devices, and there is no need for additional infrastructure investment since Place Lab relies on the existence of pre-established WiFi access points used to provide wireless connectivity. It simply requires the user to preload a database of access point coordinates. Place Lab can operate anywhere, indoors or outdoors, within the range of 802.11 access points.

Radar [1] is an earlier example of Place Lab's approach that required a calibration process for the client device. Although it achieved resolution on the order of 5m when finely calibrated it suffers from similar limitations as Place Lab.

ActiveBadge does not provide absolute coordinates like GPS or Place Lab, but rather, it provides room-level positioning within a building. ActiveBadge uses infrared-emitting *badges* that transmit unique IDs through infrared to room-aware receivers that update the position of the badge in a centralized database [11]. ActiveBadge is similar to WALRUS in that it provides room-level positioning, however unlike WALRUS, ActiveBadge is centrally managed. ActiveBadge does not grant user privacy since a centralized computing system is required to track the location of all the badges. Unlike WALRUS, which relies entirely on existing hardware, ActiveBadge requires both dedicated badges and IR receivers. Although the costs of these

items are not unreasonably high they must be purchased in large quantities and distributed appropriately throughout a building to achieve the desired coverage.

HP's CoolTown [13] adds IR transmission capability to objects/appliances of interest. Client devices hear objects' URLs and can access web pages to control the objects or find out more about them. CoolTown also requires line-of-sight between client and object and requires a database of object positions to enable localization.

The ActiveBat system is similar to the ActiveBadge system in architecture. However, ActiveBats use radio-synchronized ultrasound instead of infrared and provide a higher level of precision. ActiveBat-enabled devices emit a pulse of ultrasound when prompted via radio by the infrastructure and are localized in three dimensions within a room, by the centralized system, using measurements from time-of-flight calculations to various ultrasound receivers scattered throughout the room (usually in the ceiling) [4]. The ActiveBat infrastructure is costly both to purchase and to install (requiring detailed surveying of each receiver's position). Once Active Bat is set up, however, it is capable of locating devices with near-centimeter precision. In contrast, WALRUS provides a much more coarse-grained precision than ActiveBat, but WALRUS can also be decentralized, unlike ActiveBat, and can support a larger number of users because no coordination is required among all the clients, only among the emitters which are bounded by the infrastructure. The information concerning the location of the ActiveBat devices is managed by a central server, thus greatly reducing the privacy-friendliness of the ActiveBat system. This coordination is essential in guaranteeing efficient use of the available ultrasound bandwidth among all the client devices that must emit sound to be positioned. ActiveBats, like WALRUS must operate indoors for the ultrasound receivers to be effective.

The Cricket location system, unlike either ActiveBat or ActiveBadge, emphasizes a lack of dependence on a centralized structure to implement a complete positioning system. The Cricket system uses fixed beacons with known coordinates to emit ultrasound pulses that are used by mobile receivers to estimate position through time-of-flight calculations [8, 10]. Cricket is decentralized like WALRUS, so it preserves privacy by performing location calculations directly on the mobile clients. However, unlike WALRUS, which utilizes pre-existing hardware, Cricket requires the purchase and installation of special beacons and receivers. Once Cricket's infrastructure is established in an indoor environment, the system provides positioning precision to within $1m^2$ regions within a room (since the ultrasound does not travel through walls).

Finally, E911/E112 systems being developed for cellular phones are another ubiquitous location technology. They are being mandated by governments to help locate the originators of emergency phone calls. These systems rely on cell service providers' ability to triangulate from their cell towers to individual phones. Therefore, the infrastructure computes the location and provides it to emergency services or the user, often for a fee, thereby potentially limiting the applications that will take advantage of this capability. The precision of this approach is much less than GPS and is mandated to be of the order of 100m. Although advances in technology may enable time-of-flight measurements that will radically increase accuracy, this type of system will never be able to provide users with information about the room they are in or even the name of the floor they are on (e.g., 2nd mezzanine or 3rd level of the basement garage) as they require detailed knowledge of each building's layout and numbering scheme for floors and rooms.

Table 1 summarizes the previous discussion of existing location sensing technologies. The positioning systems WALRUS, GPS, Place Lab, Radar, ActiveBadge, Active Bat, Cricket, and E911/E112 are compared with respect to privacy, cost, precision, and operational scope.

3. IMPLEMENTATION

3.1 Design

WALRUS is a distinct point in the space of possible positioning systems. It draws different components and ideas from other positioning systems but employs a unique mixture of these elements to achieve its goals. Place Lab and Radar provide the idea of using pre-existing technology to reduce cost: Cricket and GPS provide the concept of decentralization for the purposes of maintaining privacy; Cricket and ActiveBats' use of human-inaudible ultrasound provides the medium for associating locations to devices; and ActiveBadge's room-level precision provides the inspiration for design simplicity. These concepts are combined in WALRUS to produce a low-cost, private, indoor, room-level positioning system.

Like the Cricket positioning system, WALRUS uses ultrasound beacons to provide mobile devices with the opportunity to determine their location. However,

WALRUS requires no additional hardware. Standard desktop speakers are used to transmit ultrasound and typical PDA/laptop microphones to receive it. In addition, WALRUS uses 802.11 communication as the information-bearing channel (via broadcast UDP datagrams) rather than a specialized radio.

In [6], the use of common computing hardware is used to readily transmit and receive ultrasonic signals without intrusive human obstruction. Unlike [6], WALRUS does not send data in the ultrasound signal – this radically reduces computational requirements and decreases resolution latency. The ultrasound sources do not need to be coordinated and there can be more than one to better cover larger rooms (those significantly more than 10m in any dimension).

As one of the key design considerations of WALRUS is to use preexisting hardware for positioning purposes, all major components of the system reside in software. WALRUS is implemented in Java in order to aid in the portability of the system across an array of constantly evolving mobile technologies. Two main components comprise the WALRUS system: the *server-side beacon* software and the *mobile client software*.

A server beacon runs as a desktop PC with attached speakers (most standard speakers easily produce sound in the range we are interested in – around 21KHz); it provides ultrasound for the mobile devices to receive and can send 802.11 packets to access points (APs) in the infrastructure for broadcasting (alternatively, if the PC has a wireless networking capability, it can broadcast the packet itself, but this is not a requirement). A mobile client must have an integrated microphone (such as most PDAs, laptops, and tablets as well as cell phones) that can discern the presence of an ultrasound signal (most mobile device microphones can easily do this) and must be able to receive 802.11 broadcast packets through typical WiFi cards.

The 802.11 packets are broadcast by APs and, depending on how the wireless network (WLAN) is configured, may be heard in an area much larger than that covered by a single room. This will limit the rate at which localization events can occur for each room. For client devices to hear these broadcast packets, they need to be in "monitor mode" (that is, not associated with a specific AP for a network connection). Therefore, currently, we can't localize while the user is communicating over the network and vice-versa. However, there is no technical reason this problem is not surmountable. Only minor software changes to the APs and wireless interface drivers are needed. This is consistent with a model of ubiquitous broadcast computing advocated in [12].

3.2 Server Software

When initiated, the server software begins by opening and parsing a simple file that contains a description of the room where the server is actually located. The contents of the file can easily be modified to include simple room attributes, such as the name of the room, the amenities available, who to contact in case of inquires, URLs to relevant web pages, etc. The contents of the room description file are packaged into an 802.11 datagram packet. The server periodically (with some random variation to avoid collisions with nearby servers) broadcasts the room's data packet (either directly or through an AP) simultaneously with a short audio signal at 21KHz. Of course, an appropriate speaker must be present on the server in order for it to broadcast the ultrasonic signal (a typical desktop PC with typical speakers does just fine – there are no special sound card requirements).

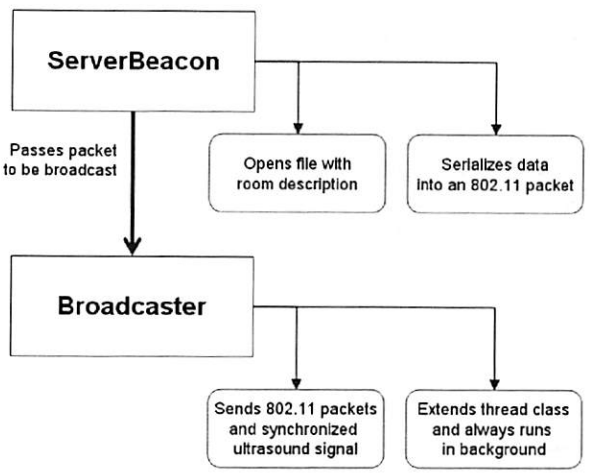


Figure 1. The ServerBeacon system runs on a desktop PC with typical speakers.

Because no data is transmitted in the ultrasound there is no need to dynamically modulate the sound wave. As a result, the ultrasonic signal can be pre-generated once and stored in any number of audio formats. After experimenting with several different methods for ultrasonic generation, we discovered that a utility called *sox*, was ideal for this purpose. Additionally, the amplitude of the audio signal is linearly increased and decreased at the beginning and end of playback, respectively. This ramping method creates a trapezoidally-shaped envelope for the amplitude of the sound and effectively band-limits the signal, avoiding the generation of audible clicks at the edges of the pulse.

Figure 1 graphically describes the hierarchical overview of the server software. In Java code, a ServerBeacon object instantiates a Broadcaster object after opening and parsing the appropriate room description file. The Broadcaster is responsible for periodically sending both the ultrasonic signal and 802.11 packet. Broadcasting time is randomized on the server to minimize the possibility of two separate servers interfering.

A further optimization, which we have not yet implemented, is to coordinate the servers so that they take explicit turns in broadcasting their data. This time-multiplexing scheme is employed by ActiveBats [4] where a round-robin schedule is used to serialize each Bat's ultrasound pulse. This approach limits the rate at which localization events can be accomplished but decreases the chance of collisions making the system more robust and more likely to make use of each ultrasound pulse. Furthermore, the system can be further optimized to exclude empty rooms, detected by motion sensors, from the schedule.

3.3 Mobile Client Software

The task of the mobile client software is to listen for 802.11 location description packets and then listen for a corresponding ultrasound pulse. Of course, the device may hear multiple 802.11 packets as RF travels through walls but it should only hear ultrasound from speakers in the same room.

Currently, the mobile client software requires the mobile device to have basic recording capabilities and either a floating point processor or floating point emulation instructions. Common mobile devices used throughout the design and development of the WALRUS system include a Dell Inspiron 8200 laptop with a 2.2GHz Mobile Pentium 4 processor and an HP iPAQ 3870 with a 206MHz StrongArm processor both running Linux operating systems. Although these are highly capable systems, we do not see obstacles to transforming our computation to fixed point arithmetic so that it can be performed on less capable devices.

While it does take longer for WALRUS to execute on mobile devices that have only floating point emulation rather than those that use a true floating point processor, we discovered that we were able to minimize the delay due to floating point emulation on an HP iPAQ 3870 by performing the instructions in a software pipelined fashion.

The mobile client component is designed to run whenever it hears an 802.11 location packet. It records audio from the microphone for enough time for a typical room size (e.g., 50-100ms can handle a good sized room of 18m maximum dimension at 25°C). It then looks for energy in the received audio in a small band around 21KHz. If there is a signal there, then it is likely the device is in the room that generated the last location information-bearing 802.11 packet. Conversely, if there is no energy at 21KHz, then it is likely that the device is in a different room. Several readings over a few seconds can quickly provide a high-confidence room-level location estimate. Note that the client device only expends energy on ultrasound detection when it hears an 802.11 location packet. The rest of the time, it performs no ultrasound calculations at all.

The MobileClient class is responsible for instantiating the correct objects to detect 802.11 packets and ultrasonic signals, as well as maintaining a probabilistic location analysis based on the positive detection of ultrasonic and datagram signals. The MobileClient ultimately provides location information to the granularity of a room inside a building based on the reception of 802.11 broadcasts and the corresponding detection of the paired ultrasound pulse.

The room information is in a form deemed appropriate by whoever setup the ServerBeacon in that room. We expect typical information to include: room number, floor, organization to whom the room belongs, phone number for facilities personnel, optionally encrypted occupant information, URLs to web pages describing aspects of the room, etc. For example, a conference room in our department could broadcast: that it is room 403; it is on the 4th floor of the Allen Center; that the building is the Paul G. Allen Center at the University of Washington; a URL to a page with links to the department's home page, building directory and floor-plan, sign-up calendar for the room; the IP address of a projector available for presentations, the IP address of a large flat panel display, and instructions on how to connect to a guest wireless network.

Also written in Java for portability, the Mobile Client software is comprised of two main components: a DatagramListener (for room information packets) and an UltrasoundListener (see Figure 2).

As already mentioned, the client can't do localization at the same time as it is communicating over the network for data transfers (e.g., web browsing). The current limitations of wireless cards and APs require two incompatible modes of operation: monitor mode for listening to broadcast packets anonymously without requiring association to an AP (we do not want to require association for reasons of preserving anonymity for clients); and infrastructure mode for two-way

networking with AP association a required part of this mode.

3.3.1 Datagram Signal Detection

The DatagramListener is a thread that continuously receives UDP packets through the 802.11 wireless protocol and forwards these messages, which contain room-identifying information, to the MobileClient. The DatagramListener initializes a multicast socket and waits until it receives a datagram packet. Upon reception of a message, the packet is time stamped and the MobileClient is notified of an available packet for interpretation.

Once a MobileClient receives notification from the DatagramListener that a message has been received, the message is added to a data structure in the MobileClient object that allows later association and probability analysis. The MobileClient then waits for positive ultrasonic detection by the UltrasoundListener. The amount of time to wait is dependent on the size of the room – this data can also be broadcast in the location information packet over 802.11 so that each room provides its own customized timeout interval.

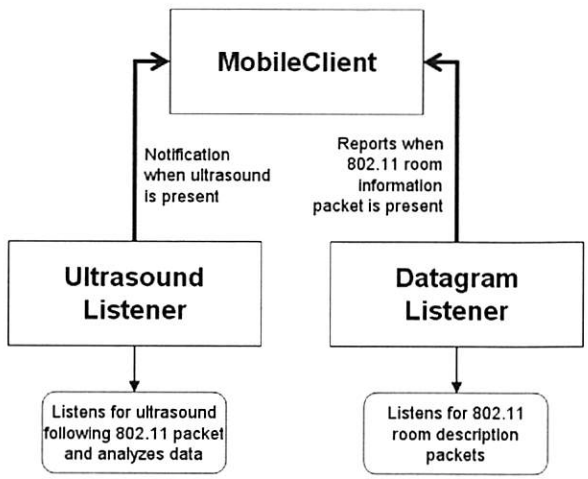


Figure 2. A MobileClient consists of two main parts: an UltrasoundListener and a DatagramListener.

3.3.2 Ultrasonic Signal Detection

The UltrasoundListener thread continuously records audio in the specified interval after the 802.11 datagram is received and notifies the MobileClient when it detects ultrasound. The MobileClient object attempts to correlate the datagram messages with the ultrasound detections in order to find the best match that will provide an accurate room determination. In the Java code, an UltrasoundListener is a thread that acquires an available microphone interface to be used for recording ultrasonic signals and instantiates a pool of UltrasoundAnalyzer threads, which are later used in calling and evaluating the captured audio signals.

The UltrasoundAnalyzer thread has a method called available() which is used by the UltrasoundListener object to determine whether there are available analyzer threads ready to receive data. Analysis of recorded audio data begins after data is available and the analyze() method of an UltrasoundAnalyzer is called. Once the analysis is complete, the UltrasoundAnalyzer thread signals the UltrasoundListener object, records the time if ultrasound was detected and becomes available to analyze another set of data.

The UltrasoundDetector object is responsible for calling an appropriate digital signal processing algorithm. WALRUS currently uses the Goertzel algorithm because, similar to a narrow bandpass filter, it is useful when analyzing a signal for energy in a small band centered on a particular frequency. Additionally, the Goertzel algorithm has several optimizations that allow for very quick computations [2].

The UltrasoundDetector object instantiates three instances of the Goertzel algorithm in order to compare the relative magnitudes of the desired ultrasonic frequency to two adjacent frequencies: one above and one just below 21KHz. Since white noise contains a wide variety of similar frequencies, we can largely eliminate the possibility of detecting unwanted ultrasonic sounds by ensuring that the relative magnitude of the 21KHz frequency is much greater than the relative magnitudes of signals with frequencies just above and below 21KHz. Prior to using three instances of the Goertzel algorithm, false positives were often detected when the doors shut or the detecting microphone heard a whistle or jingling keys. However, once three Goertzel algorthim instances were used by the UltrasoundDetector, no false positives were detected during any of the remaining tests.

3.3.3 Determining the Location

The MobileClient manages a data structure that keeps a running history of the 802.11 packets received. After the MobileClient class has been notified of an ultrasonic detection, analysis must be done to match the ultrasonic detection with an 802.11 packet and derive a probability that the two signals are coupled.

Upon detecting ultrasound, the data structure storing 802.11 packets is examined to determine if at least one 802.11 packet has arrived within the Maximum

Detection Time. The Maximum Detection Time defines the time interval to be examined. It is determined from the 802.11 packet reception time and the amount of time it specifies to wait for its corresponding room. If only one 802.11 packet is received within the Maximum Detection Time period, then this 802.11 packet is easily associated with the ultrasound signal. If more than one 802.11 packet is present then the ADT, the Average Detection Time, is used to make a decision. The ADT is a value derived from historical data for the current room. It provides some hysteresis to changes in room estimates.

Figure 3 describes the process of associating an ultrasonic signal and 802.11 datagram. In the figure, 802.11 packets from Rooms A, B, and C are considered because they all fall within the Maximum Detection Time range. That is, they may have been the packets associated with the ultrasound signal that was detected. We consider the difference in time between the detected ultrasound and each of the packets received. In the case of Figure 3, the packet for Room C has the smallest difference between its arrival time and the ultrasound detection. We then compare these differences against the current ADT. Since the difference between the ultrasound detection event and the packet for Room B is the closest is value to the current ADT, we associate this ultrasound signal with Room B. This may be wrong and is likely to be corrected when the next round of packets are broadcast as they are probabilistically very unlikely to cause similar confusion. Finally, the ADT for the current room is updated by averaging in this latest detection time difference.

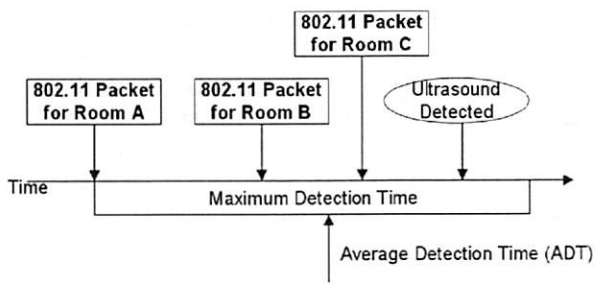


Figure 3. After ultrasound detection, the previous 802.11 messages received are analyzed to determine the most likely match using an average detection time (ADT) derived from recent history.

Once the ultrasonic signal is associated with an 802.11 signal, a probability that the association is correct needs to be determined. Since a history of messages is stored in a data structure, a weighted value is assigned to each element of the history based on how recently the message was added to the history, and whether a sequence of messages exists from a particular room. For example, if three messages are received consecutively from Room A, then not only will Room A receive an advantageous weight value over older messages in the data structure, but it will receive an additional weight since a sequence of consecutive messages was seen from a particular room. The weighted values could be used by an application in a variety of ways, for example, to fetch information about several rooms rather than just one and let the user disambiguate. The value literally describes WALRUS' confidence level that a mobile device is within a particular room.

4. EVALUATION

WALRUS has proven itself to be a robust system for determining room location. Our tests have shown that it provides accurate and consistent localization with room-level resolution. Anyone carrying a mobile device enabled with WALRUS in a WALRUS-enabled space can walk from room to room and determine their location via the mobile device.

Since the WALRUS-enabled mobile device is able to determine its room location in terms meaningful to a person, it can use this information to provide details about nearby services to the user. An external application could be constructed containing a database of known services and their locations on a virtual map. Once the mobile device learns its position on the virtual map, it would be able to determine what services are nearby and provide navigation descriptions that include hallways, intersections, and other landmarks that can be much more meaningful and effective for people as opposed to simple Cartesian distances.

Attempting to use readily-available technology as a foundation for our system has also proven to be a valid endeavor. Anyone who wants to equip an office building with location technology needs only to install and run our server beacon software on a speaker-equipped desktop system in each room of interest, utilize likely pre-existing WiFi access points, and install and run our mobile client software on a microphone-equipped mobile device of choice. No additional setup or hardware is required for our system.

As a tradeoff to its high degree of reliability, WALRUS suffers slightly from slow performance. Most of this delay is used to increase accuracy as well as reliability. For example, the period of time between server beacon broadcasts must be large enough so that the chances of

nearly simultaneously received messages are reduced. For a typical office scenario, we can assume rooms on the order of $25m^2$ and that the ultrasound may travel twice the length of the room to ensure reflections die out. This 10m distance is traversed in approximately 30msec at 20^{oC}. If we assume a 10msec ultrasound pulse, this allows us to realize 25 (1/(.030+.010)) localization events per second if we schedule each room to fire its beacon packet in turn. If we assume that a typical WiFi AP has a range of 50m and, therefore, covers approximately 100 of our $25m^2$ rooms, then we can localize every 4 seconds. Even though this estimate is somewhat pessimistic as significant space is occupied by hallways and many APs have reduced range through several walls, we feel it is a reasonable rate to expect from WALRUS in practice.

A history of recent room determinations can be kept on the client and used to calculate relative probabilities for room locations so that incorrect room determinations do not immediately cause the mobile device to believe that it is in a different room. However, this increases the amount of time it takes for the mobile client to change its belief in a given room location when moving between rooms because multiple identical room determinations must occur before the mobile client indicates a large enough confidence that it has changed rooms. Experimentation is needed to determine the best values for some of these parameters.

The system as it currently exists is not well suited for interactive map applications since the mobile device's position on a virtual map would be perceptibly skewed in time from its actual physical location. It could take several seconds for the mobile device's position on the virtual map to update once the mobile device crosses a room boundary. WALRUS is better suited for non-interactive location awareness in devices that move at human speeds from room to room or that do not need immediate location resolving. For example, WALRUS could be used to help mobile devices connect to infrastructure resources in a particular room (e.g., a smart conference room). WALRUS could also benefit someone who occasionally moves from office to office during the day but who enjoys the benefits of certain location-aware applications on his mobile device. WALRUS is particularly useful for mobile workers who reach a new location and need access to local information.

Though relatively slow, WALRUS does offer accurate ultrasound detection, the critical element for an accurate room determination. According to our tests, the mobile client, consisting of a Dell Inspiron 8200 laptop with built-in microphone, is able to correctly detect ultrasound with little error across distances up to approximately 13 meters. WALRUS is able to achieve this because there is no data to decode in the ultrasound signal. In [3], it was shown that is was not possible to transmit data reliably at 18.4KHz over standard speakers/microphones. In [6], only 95% of packets were decoded correctly over ultrasound at 21KHz even over distances as short as 1-1.5m. Also, in [6], it was shown that it took approximately 15 seconds to transmit an 8-bit room identifier over ultrasound because the signal-to-noise ratio (SNR) was so low. WALRUS does not suffer from these problems because it only looks for an energy peak at 21KHz and is not concerned with a long-lived transmission for decoding data. WALRUS ultrasound pulses can be as short as a few milliseconds thus reducing latency dramatically.

In order to test our system's ultrasound detection reliability, we used our server beacon code (running on a Dell Inspiron 600m notebook computer) to broadcast a 10 ms ultrasound pulse at regular intervals from a standard, cheap pair of desktop speakers (Altec Lansing ADA215). We then used our mobile laptop device (Dell Inspiron 600 notebook computer) to execute our mobile client software at varying distances from the server beacon's speakers. We examined the times at which ultrasound was received on the laptop and tried to correlate these with the regular intervals at which ultrasound was being broadcast. Long periods of time between ultrasound detections were noted as missed detections and short periods of time between ultrasound detections were noted as false positive detections. This set of tests was performed in a computer laboratory environment, filled with both the ambient noise of people working and various shelves and racks that could interfere with the transmission of ultrasound – an environment not advantageous to our approach (see Figure 4 for a photograph of the lab in which the tests were conducted – notice the large metal benches providing many surfaces for ultrasound reflections). The results of these tests are shown in Figure 5.

The percentage of the time that ultrasound was detected when ultrasound was actually present, as well as the percentage of time that ultrasound was detected when ultrasound was not present, is given at various distances from the server beacon. The graph shows that correct ultrasound detection occurs nearly one hundred percent of the time up until about 10 meters away from the server beacon, it drops off sharply after that point. However, it is anticipated that most rooms used to contain WALRUS server beacons will not be so large; such expansive rooms would most likely require multiple sound sources for the same room information. The chart also shows that false positive detections of

ultrasound occur anywhere from zero to three percent of the time and become more noticeable only when the distances are greater. It is possible that the false positive ultrasound detections are caused by delayed ultrasonic sound waves arriving at the laptop through different paths than the direct one traversed by most of the ultrasonic energy. As these findings show, WALRUS is effective at detecting ultrasound at distances up to about 10m – a good distance for most rooms in an office building.

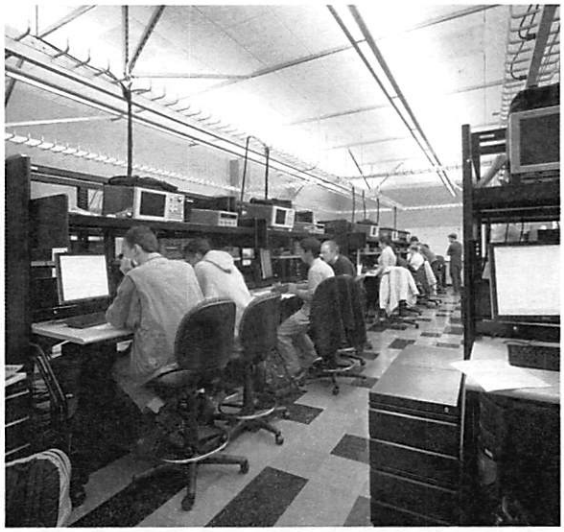


Figure 4. Laboratory environment for our initial tests.

We also performed a variety of trials to determine if common environmental noise would cause the mobile client to detect ultrasound when it was not sent by the server. For these tests we used a typical office in our building that includes whiteboard partitions, book shelves, and many desk surfaces (see Figure 6). The office is 5m by 7m.

In the first trial, the mobile client was placed next to three people in an office having a conversation at normal volume. In the second trial, a variety of MP3- and Ogg Vorbis- encoded music, including Jazz, Rhythm and Blues, Alternative, Classic Rock, and Techno, were played through the server's speakers placed 5ft from the client. In the third trial, the client was left running over two days in an office, while events such as doors slamming, keys jingling, people talking, and cell phones ringing occurred. No ultrasound was detected during any of the trials.

To see whether loud noise affects ultrasound detection, the above musical selections were played on a separate speaker next to the client while the server generated ultrasound pulses. No difference in accuracy of ultrasound detection was observed. The results were not significantly different even in the presence of vertical whiteboards extending to 5 feet in height. The parameter that most affected the results was the orientation of the speakers to the client. The best case is when the client is directly facing the speakers, there is a noticeable drop-off in detection precision after 10m when the client is at 90° to the speakers. However, this distance is greater than the dimensions of the rooms of interest to us. This gives us confidence that our approach can be quite robust and highly accurate in typical office environments.

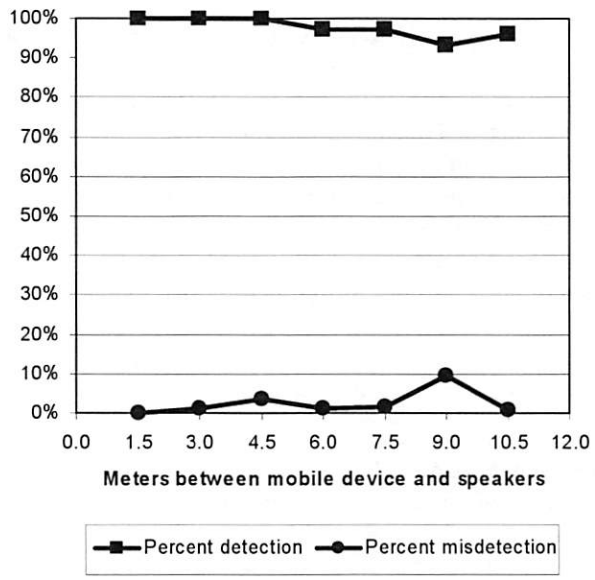


Figure 5. Results of ultrasound detection experiments.

We also did tests that varied the volume of the speakers to see if we could control "leakage" of the ultrasound out of an open doorway. We tried 5 different volume positions corresponding to Windows XP's 20%, 40%, 60%, 80%, and 100%. We felt this was a reasonable, if imprecise experiment as this is the setting most users will adjust on their office PCs and we wanted to reflect that is likely to happen in practice. Within the office, ultrasound detections were almost perfect at all levels for distances up to 5m. We noticed a drop in detection accuracy (down to 88%) at the 20% setting at 5m distance. Leakage out of the office door is shown in the two parts of Figure 7. On the left, the speakers are oriented are far from the door and the two curves show

detection of 20% and 40% settings. On the right, the speakers are near the door and the volume is set to 20%.

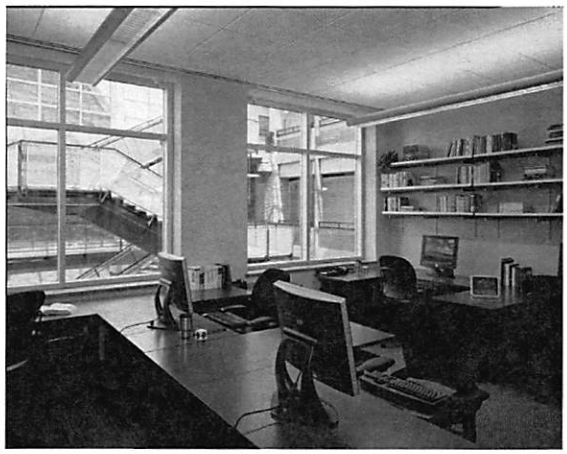


Figure 6. Office environment similar to that used for our tests.

We can see that it is practical to have the speakers set at a low volume in most office settings so that leakage is confined to the area immediately outside the door. Even if the client is outside the office, this location determination is likely to be appropriate for hallways.

The certainty with which WALRUS makes room determinations is dependent on the number of room location messages that the system is receiving. Since WALRUS attempts to correlate the times that it received ultrasound with the times that it received room location messages, the occurrence of several room location messages in close time proximity makes it difficult for the system to choose the best pairing of 802.11 messages and ultrasound detections.

In order to make quantitative assessments about WALRUS' ability to correctly determine room locations, we set up varying numbers of server beacons to broadcast different room messages. Each of these server beacons was located in the same room and used the same AP to broadcast its location description packet, but only one of the beacons was equipped with standard desktop speakers for broadcasting ultrasound. Every time the mobile client correctly determined that it was in the room described by the room message from the beacon with speakers, this was noted as a correct room determination. Every time the mobile client determined that it was in the room described by the room message from one of the other beacons, this was noted as an incorrect room determination. The results of these tests with varying numbers of server beacons are given in Figure 8.

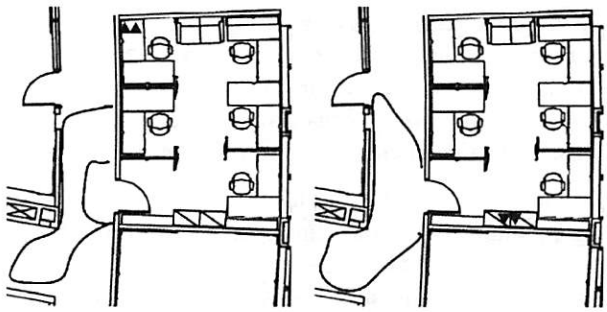


Figure 7. Leakage of ultrasound past office door for two different positions of the speakers (shown as two triangles).

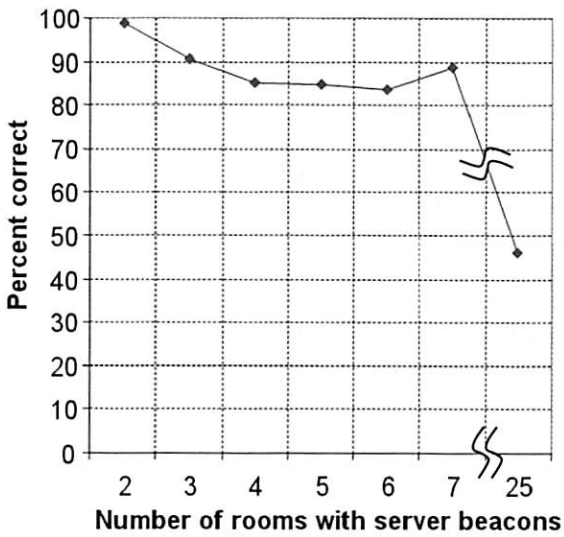


Figure 8. Room determination reliability with varying numbers of server beacons.

This chart shows the percentage of time that the mobile client correctly correlated ultrasound detection with the room message being broadcast by the beacon with speakers. As seen in the chart, WALRUS has near-perfect location determination with only two server beacons running – as one might expect, the chance of collisions is low. However, this accuracy drops as the number of beacons using the same AP increases. With six server beacons running, the mobile client is only able to correctly determine room location about 84% of the time. In order to view the trend of decreasing accuracy for larger numbers of server beacons, this test was also run with 25 server beacons broadcasting different room messages. As the chart shows, this large

number of beacons within range of the mobile client causes the accuracy in room determination to drop below 50%. Thus, it can be seen that WALRUS's ability to correctly correlate ultrasound with received room messages drops steadily as the number of running server beacons within range of the mobile client increases. Use of historical data provides some smoothing that corrects many of these errors at the cost of some further latency in room determination.

In order for WALRUS to be an effective indoor positioning system, the number of location description messages received by mobile clients from nearby server beacons can be reduced fairly easily by limiting the server beacons to broadcast UDP packets only through their local WiFi AP so that the number of beacons heard in a region is reduced to the minimum. In fact, we envision a basic device that can be left in a room (or connected to a PC) with a purposely short-range WiFi transceiver and an ultrasound-optimized speaker. Such a package could easily cost on the order of USD20 and be very trivial to configure – it simply needs a room information packet and transmission period with dynamic information available from a central server that sends packets to all of these units. Each can pick out the information relevant to its room. In this manner, we would expect to reduce the number of server beacons that can be confused by a client to only a few. Two or three localization events (8 to 12 seconds) then provide a high probability of a correct room determination. More importantly, such a specialized device obviates the need to keep a PC running in every room and could be made small enough to just fit into an electrical outlet, light switch, or even light bulb socket.

The WALRUS positioning system suffers slightly from slow performance, but it is highly reliable and accurate in office-scale environments. It is useful for all the situations in which we initially anticipated it being used. WALRUS is definitely not perfect. There are several tuning parameters to be experimented with and there is still a great deal of work that could be done to make it faster and more expandable to larger-scale environments with more densely packed rooms.

5. FUTURE WORK

Several options exist for future work to improve the WALRUS system. One key feature that still needs to be examined is to determine how well WALRUS works when it is deployed in a large-scale setting. Due to time constraints, we have only been able to extensively test WALRUS in a small-scale environment consisting of a few rooms although we have run tests with many more server beacons. It is believed that scaling WALRUS should be possible with very little additional effort. Since no specialized hardware is needed, only installation of the WALRUS software is needed to begin exploring the effects of using the system in an environment with many rooms and corresponding servers.

Multiple avenues could also be taken to improve ultrasonic detection. Currently, WALRUS uses floating point instructions within the Goertzel algorithm to analyze the data recorded from a mobile device. It is possible to implement the Goertzel algorithm using fixed point arithmetic; this optimization would eliminate the use of costly floating point emulation instructions and radically speed up analysis leading to an even lower-power implementation that could be ported to an even wider range of devices – our intention is to ultimately have WALRUS running on a wrist-watch. Additionally, we believe alternate signal analysis methods might be developed that are less computationally intensive than Goertzel, but still appropriately for WALRUS.

Currently, WALRUS has a very high level of accuracy with room level precision. We have also discovered that we are often able to quickly interpret ultrasonic signals across very large rooms. One experiment that should be considered is to determine what amount of accuracy, if any, can be sacrificed in order to obtain a higher level of precision. By using time-of-flight or alternate methods it may be possible to accurately narrow the system's estimation of a user's location within a room to several square feet. This would bring more Cricket-like functionality to WALRUS.

Integrating WALRUS with Place Lab could help our disambiguation of multiple WiFi room information packets. Place Lab can be used to determine a coarse 3-D location that can be used to filter the room location packets for only those that are truly possible and eliminate outliers from rooms further away. This would allow us to dramatically increase the rate of localization events as rooms within the same AP's range could generate ultrasound pulses in parallel.

Finally, we are developing a set of building-scale applications that use room-level location information. Many were described as examples throughout this paper. We also plan to integrate WALRUS with our Ubiquitous Broadcast Computing infrastructure (UBC) that uses similar concepts of WiFi broadcasts to distribute information to mobile users [12].

6. CONCLUSIONS

There exist a wide array of systems that provide a mobile device with information that can be used to determine that device's location. After analyzing several existing location aware systems, we decided it was possible to design a system that uses only pre-existing hardware, but is still capable of providing a user with enough information to derive their location with room-level granularity. We borrowed key design features from systems such as Cricket and ActiveBadge and were able to incorporate elements of those designs into WALRUS and created a system where no specialized and costly hardware is needed. The resulting system takes advantage of the built-in microphones available on many mobile devices and the existing speakers common to most desktop systems to create a location system that allows a mobile device to privately detect ultrasonic signals and correlate these signals with WiFi packets broadcast by a desktop system stationed within a given room (through APs in the infrastructure). We found this system to be an accurate, manageable, and extremely affordable solution for determining the room in which a device is located within environments such as large office buildings. Many simple adjustments can be made during future implementations, which could increase the overall performance and precision of the system.

7. ACKNOWLEDGEMENTS

We would like to thank the many individuals who have provided us with support and guidance throughout the duration of this project, especially, Tom Anderl, Todd Drullinger, Jong Hee Kang, and James Scott.

8. REFERENCES

[1] P. Bahl and V. N. Padmanabhan. RADAR: An In-Building RF-Based User Location and Tracking System, Proceedings of IEEE INFOCOM 2000, Vol. 2, pp. 775-784, March 2000.

[2] K. Banks. The Goertzel Algorithm, Embedded.com, http://www.embedded.com/story/OEG20020819S0057, August 28, 2002.

[3] V. Gerasimov and W. Bender. Things that talk: Using sound for device-to-device and device-to-human communication. IBM Systems Journal Vol 39, No 3&4, pp 530-546, 2000.

[4] A. Harter, et al. The Anatomy of a Context-Aware Application, 5th ACM International Conference on Mobile Computing and Networking (Mobicom99), ACM Press, New York, pp. 59-68, August 1999.

[5] J. Hightower and G. Borriello. Location Systems for Ubiquitous Computing, IEEE Computer, vol. 34, no. 8, pp. 57-66, IEEE Computer Society Press, August 2001.

[6] A. Madhavapeddy, D. Scott, R. Sharp. Context-aware computing with sound, 5th International Conference on Ubiquitous Computing (UbiComp 2003), September 2003.

[7] A. K. L. Miu. Design and Implementation of an Indoor Mobile Navigation System, SM Thesis, Massachusetts Institute of Technology, January 2002.

[8] N. B. Priyantha, A. Chakraborty, H. Balakrishnan. The Cricket Location-Support System, 6th ACM International Conference on Mobile Computing and Networking (Mobicom00), Boston, MA, August 2000.

[9] B. Schilit, A. LaMarca, G. Borriello, et. al. Challenge: Ubiquitous Location-Aware Computing and the Place Lab Initiative, 1st ACM International Workshop on Wireless Mobile Applications and Services on WLAN (WMASH 2003), San Diego, CA, September 2003.

[10] A. Smith, H. Balakrishnan, M. Goraczko. Tracking Moving Devices with the Cricket Location System, 2nd International Conference on Mobile Systems, Applications and Services (Mobisys 2004), Boston, MA, June 2004.

[11] R. Want, et al. The Active Badge Location System, ACM Transactions on Information Systems, pp. 91-102, January 1992.

[12] J. H. Kang, G. Borriello. Ubiquitous Computing using Wireless Broadcast. 6th IEEE Workshop on Mobile Computing Systems & Applications (WMCSA 2004), Lake District, UK, December 2004.

[13] T. Kindberg, J. Barton, et. al. People, Places, Things: Web Presence for the Real World, 3rd IEEE Workshop on Mobile Computing Systems & Applications (WMCSA2000), Monterey, CA, December 2000.

The Horus WLAN Location Determination System

Moustafa Youssef and Ashok Agrawala
Department of Computer Science
University of Maryland
College Park, Maryland 20742
{*moustafa, agrawala*}*@cs.umd.edu*

Abstract

We present the design and implementation of the *Horus* WLAN location determination system. The design of the *Horus* system aims at satisfying two goals: high accuracy and low computational requirements. The *Horus* system identifies different causes for the wireless channel variations and addresses them to achieve its high accuracy. It uses location-clustering techniques to reduce the computational requirements of the algorithm. The lightweight *Horus* algorithm helps in supporting a larger number of users by running the algorithm at the clients.

We discuss the different components of the *Horus* system and its implementation under two different operating systems and evaluate the performance of the *Horus* system on two testbeds. Our results show that the *Horus* system achieves its goal. It has an error of less than 0.6 meter on the average and its computational requirements are more than an order of magnitude better than other WLAN location determination systems. Moreover, the techniques developed in the context of the *Horus* system are general and can be applied to other WLAN location determination systems to enhance their accuracy. We also report lessons learned from experimenting with the *Horus* system and provide directions for future work.

1 Introduction

Horus is an RF-based location determination system. It is currently implemented in the context of 802.11 wireless LANs [25]. The system uses the signal strength observed for frames transmitted by the access points to infer the user location. Since the wireless cards measure the signal strength information of the received frames as part of their normal operation, this makes the *Horus* system a software solution on top of the wireless network infrastructure. There are two classes of WLAN location determination systems: client-based and infrastructure-based. Both have their own set of applications. *Horus* is currently implemented as a client-based system. A large class of applications [10], including location-sensitive content delivery, direction finding, asset tracking, and emergency notification, can be built on top of the *Horus* system.

WLAN location determination is an active research area [5, 6, 8, 9, 12, 13, 15, 17, 20–22, 29–31, 33, 34]. WLAN location determination systems usually work in two phases: an *offline* training phase and an *online* location determination phase. During the offline phase, the system tabulates the signal strength received from the access points at selected locations in the area of interest, resulting in a so-called *radio map*. During the location determination phase, the system use the signal strength samples received from the access points to "search" the radio map to estimate the user location.

Radio-map based techniques can be categorized into two broad categories: deterministic techniques and probabilistic techniques. *Deterministic techniques* [5, 6, 22] represent the signal strength of an access point at a location by a scalar value, for example, the mean value, and use non-probabilistic approaches to estimate the user location. For example, in the *Radar* system [5, 6] the authors use nearest neighborhood techniques to infer the user location. On the other hand, *probabilistic* techniques [8, 9, 13, 17, 20, 21, 29–31, 33, 34] store information about the signal strength distributions from the access points in the radio map and use probabilistic techniques to estimate the user location. For example, the Nibble system [8, 9] uses a Bayesian Network approach to estimate the user location.

The *Horus* system lies in the probabilistic techniques category. The design of the *Horus* system aims at satisfying two goals: high accuracy and low computational requirements. The *Horus* system identifies different causes for the wireless channel variations and addresses them to achieve its high accuracy. It uses location-clustering techniques to reduce the computational requirements of the algorithm. The lightweight *Horus* algorithm allows it to be implemented in energy-constrained devices. This

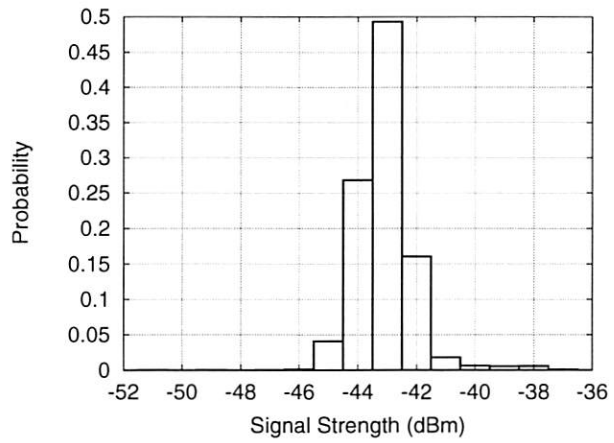


Figure 1: An example of the normalized signal strength histogram from an access point.

non-centralized implementation helps in supporting a larger number of users. In this paper, we present the different components of the *Horus* system and show how they work together to achieve its goals. We discuss our *Horus* implementation under two different operating systems and evaluate its performance on two different indoor testbeds.

The rest of the paper is structured as follows: in the next section, we describe the different causes of variations in the wireless channel. In Section 3 we present the different components of the *Horus* system that deal with the noisy characteristics of the wireless channel. We present the results of testing the *Horus* system on two different testbeds in Section 4. Section 5 presents our experience while building the *Horus* system. In Section 6 we discuss related work. Finally, Section 7 concludes the paper and provides directions for future work.

2 Wireless Channel Characteristics

In this section, we identify the different causes of variations in the wireless channel quality and how they affect the WLAN location determination systems. We are mainly concerned with the variations that affect the received signal strength. We start by describing our sampling process. Then, we categorize the variations in the wireless channel as temporal variations and spatial variations. We performed all the experiments in this section in a typical office building, measured during the day when people are around.

2.1 Sampling Process

A key function required by all WLAN location determination systems is signal-strength sampling. We used a Lucent Orinoco silver network interface card (NIC) supporting up to 11 Mbit/s data rate [3]. The *Horus* system is implemented under both the Linux and Windows operating systems.

For the Linux OS, we modified [1] the *Lucent Wavelan* driver so that it returns the signal strength of probe response frames received from all access points in the NIC range using active scanning [25]; our driver was the first to support this feature.

The scanning process output is a list of the *MAC* addresses of the access points associated with the signal strength observed in this scan (through probe response frames). Each scan's result set represents a sample.

We also developed a wireless API [1] that interfaces with any device driver that supports the wireless extensions [2]. The device driver and the wireless API have been available for public download and have been used by others in wireless research.

For the Microsoft Windows operating system, we used a custom-built NDIS driver to obtain the signal strength from the wireless card (using active scanning). This gives us more control over the scanning process as described in Section 5.

We now describe the different causes of variations in a wireless channel. We divide these causes into two categories: temporal variations and spatial variations.

2.2 Temporal variations

This section describes how the wireless channel changes over time when the user is standing at a fixed position.

2.2.1 Samples from one access point

We measured the signal strength from a single access point over a five minute period. We took the samples one second apart for a total of 300 samples. Figure 1 shows the normalized histogram of the received signal strength. In our experience, the histogram range can be as large as 10 dBm or more. This time variation of the channel can be due to changes in the physical environment such as people moving about [23].

These variations suggest that the radio map should reflect this range of values to increase the accuracy. Moreover, during the online phase, the system should use more than one sample in the estimation process to have a better estimate of the signal strength at a location.

2.2.2 Samples Correlation

Figure 2 shows the autocorrelation function of the samples collected from **one access point** (one sample per second) at a fixed position. The figure shows that the autocorrelation of consecutive samples ($lag = 1$) is as high as 0.9. This high autocorrelation is expected as over a short period of time the signal strength received from an access point at a particular location is relatively stable (modulo the changes in the environment discussed in the previous section).

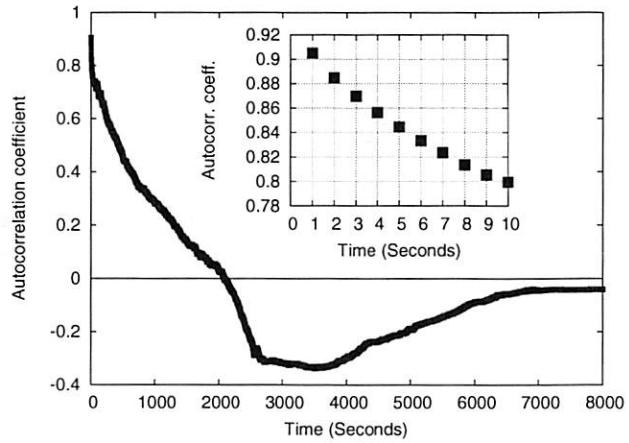


Figure 2: An example of the autocorrelation between samples from an access point (one sample per second). The sub-figure shows the autocorrelation for the first 10 seconds.

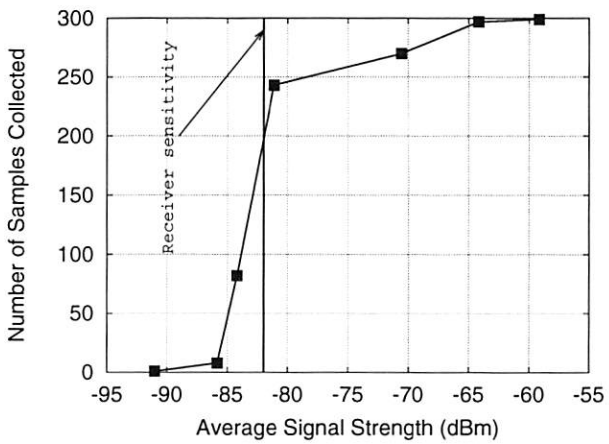


Figure 3: Relation between the average signal strength of an access point and the percentage of samples received from it during a 5-minute interval.

This high autocorrelation value has to be considered when using multiple samples from one access point to enhance accuracy. Assuming independence of samples from the same access point leads to the undesirable result of degraded system performance as the number of samples is increased (as explained in Section 4) as in a typical WLAN environment samples from the same AP are highly correlated.

2.2.3 Samples from different access points

We performed an experiment to test the behavior of access points with different average signal strength at the same location. During this experiment, we sampled the signal strength from each access point at the rate of one sample per second. Figure 3 shows the relation between the average signal strength received from an access point and the percentage of samples we receive from it during a period of 5 minutes. The figure shows that the number of samples collected from an access point is a monotonically increasing function of the average signal strength of this access point. Assuming a constant noise level, the higher the signal strength, the higher the signal to noise ratio and the more probable it becomes that the 802.11b card will identify the existence of a frame. The sharp drop at about -81 dBm can be explained by noting that the receiver sensitivity (minimum signal power required to detect a frame) for the card we used was -82 dBm.

2.3 Spatial characteristics

These variations occur when the receiver position is changed. We further divide these variations into large-scale variations and small-scale variations.

2.3.1 Large-Scale Variations

Figure 4 shows the average signal strength received from an access point as the distance from it increases. The signal strength varies over a long distance due to attenuation of the RF signal.

Large-scale variations are desirable in RF-based systems as they lead to changing the signature stored in the radio map for different locations and, hence, better differentiation between these locations.

2.3.2 Small-Scale Variations

These variations happen when the user moves over a small distance (order of wavelength). This leads to changes in the average received signal strength. For the 802.11b networks working at the 2.4 GHz range, the wavelength is 12.5 cm and we measure a variation in the average signal strength up to 10 dBm in a distance as small as 7.6 cm (3 inches) (Figure 5).

Dealing with small-scale variations is challenging. To limit the radio map size and the time required to build the radio map, selected radio map locations are typically placed more than a meter apart. This means that the radio map does not capture small-scale variations leading to decreased accuracy in the current WLAN location systems.

In the next section, we indicate how the *Horus* system handles these temporal and spacial variations.

3 The *Horus* System

In this section, we present the different components of the *Horus* system.

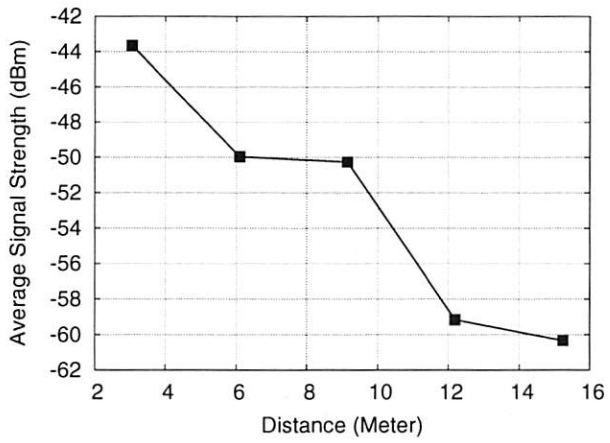


Figure 4: Large-scale variations: Average signal strength over distance.

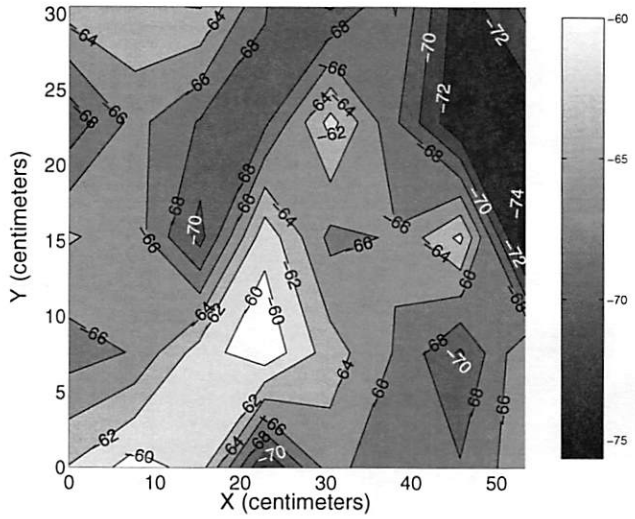


Figure 5: Small-scale variations: Signal strength contours from an AP in a 30.4 cm (12 inches) by 53.3 cm (21 inches) area.

3.1 Overview

Figure 6 shows the overall system. The *Horus* system works in two phases:

1. *Offline phase*: to build the radio map, cluster radio map locations, and do other preprocessing of the signal strength models.

2. *Online Phase*: to estimate the user location based on the received signal strength from each access point and the radio map prepared in the offline phase.

The radio map stores the distribution of signal strength received from each access point at each sampled location. There are two modes for operation of the *Horus* system: one uses non-parametric distributions and the other uses parametric distributions.

The *Clustering* module is used to group radio map locations based on the access points covering them. Clustering is used to reduce the computational requirements of the system and, hence, conserve power (Section 3.7).

The *Discrete Space Estimator* module returns the radio map location that has the maximum probability given the received signal strength vector from different access points (Section 3.3).

The *Correlation Modelling and Handling* modules use an autoregressive model to capture the correlation between consecutive samples from the same access point. This model is used to obtain a better discrete location estimate using the average of n correlated samples (Section 3.4).

The *Continuous Space Estimator* takes as an input the discrete estimated user location, one of the radio map locations, and returns a more accurate estimate of the user location in the continuous space (Section 3.5).

The *Small-Scale Compensator* module handles the small-scale variation characteristics of the wireless channel (Section 3.6).

We start by laying out the mathematical framework for the approach then give details about different components of the system.

3.2 Mathematical Model

Without loss of generality, let $\mathbb{X}$ be a 2 dimensional physical space. At each location $x \in \mathbb{X}$, we can get the signal strength from k access points. We denote the k-dimensional signal strength space as $\mathbb{S}$. Each element in this space is a k-dimensional vector whose entries represent the signal strength readings from different access points. We denote samples from the signal strength space $\mathbb{S}$ as s. We also assume that the samples from *different* access points are independent.

The problem becomes, given a signal strength vector $s = (s_1, ..., s_k)$, we want to find the location $x \in \mathbb{X}$ that maximizes the probability $P(x/s)$.

In the next section, we assume a discrete $\mathbb{X}$ space. We discuss the continuous space case in Section 3.5.

3.3 Discrete Space Estimator

During the offline phase, the *Horus* system estimates the signal strength histogram for each access point at each location. These histograms represent the *Horus* system's radio map. Now consider the online phase. Given a signal strength vector $s = (s_1, ..., s_k)$, we want to find the location $x \in \mathbb{X}$ that maximizes the probability $P(x/s)$,

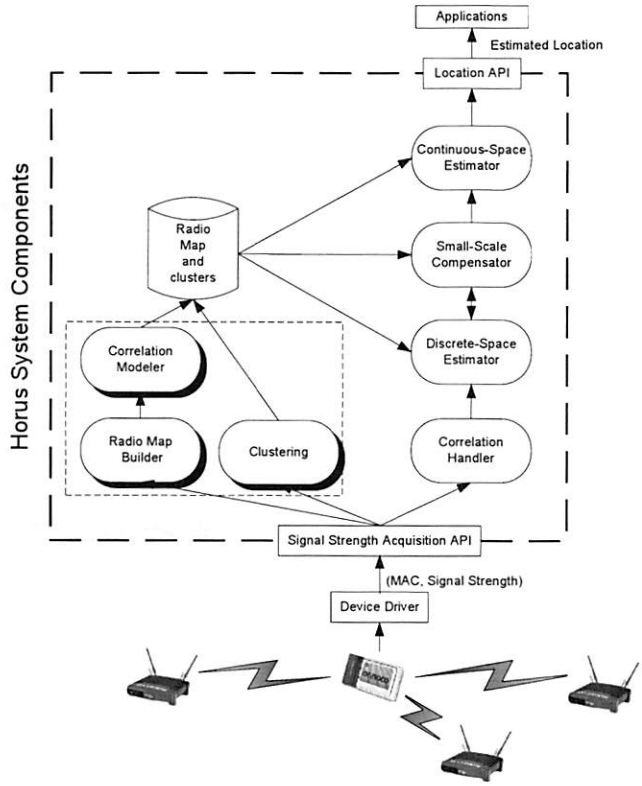


Figure 6: *Horus* Components: the arrows show information flow in the system. Shadowed blocks represent modules used during the offline phase.

i.e., we want

$$\text{argmax}_x[P(x/s)] \tag{1}$$

Using Bayes' theorem, this can be shown to be equivalent to [33]:

$$\text{argmax}_x[P(x/s)] = \text{argmax}_x[P(s/x)] \tag{2}$$

$P(s/x)$ can be calculated using the radio map as:

$$P(s/x) = \prod_{i=1}^{k} P(s_i/x) \tag{3}$$

The signal-strength histogram can be approximated by a parametric distribution such as the Gaussian distribution. We compare the performance of the discrete-space estimator based on the parametric and non-parametric distributions in the Section 4.

3.4 Correlation Handling

To account for the temporal signal-strength variations, it is important to average multiple signal strength samples from the same access point. As we showed in Figure 2, the autocorrelation of successive samples collected from one access point is as high as 0.9. Assuming independence of samples from the same access point leads to the undesirable result of degraded system performance as the number of averaged samples is increased (as we demonstrate below, in Section 4).

3.4.1 Mathematical model

We use an autoregressive model to capture the correlation between different samples from the same AP.

Let s_t be the *stationary* time series representing the samples from an access point, where t is the discrete time index. s_t can be represented as a *first order* autoregressive model [7] as:

$$s_t = \alpha s_{t-1} + (1-\alpha)v_t \quad ;0 \leq \alpha \leq 1 \tag{4}$$

where v_t is a noise process, independent of s_t, and α is a parameter that determines the degree of autocorrelation of the original samples. Moreover, different samples from v_t are independent and identically distributed (i.i.d.).

The model in Equation 4 states that the current signal strength value (s_t) is a linear aggregate of the previous signal strength value (s_{t-1}) and an independent noise value (v_t). The parameter α gives flexibility to the model as it can be used to determine the degree of autocorrelation of the original process. For example, if α is zero, the samples of the process s_t are i.i.d.'s, whereas if α is 1 the original samples are identical (autocorrelation=1).

Assuming that the signal strength distribution of samples from an access point is Gaussian with mean μ and variance σ^2, we have shown in [29, 31] that the distribution of the average of n correlated samples is a Gaussian distribution with mean μ and variance given by:

$$\frac{1+\alpha}{1-\alpha}\sigma^2 \tag{5}$$

3.4.2 Correlation modeler

The purpose of the correlation modeler component is to estimate the value of α in the autoregressive model and to pre-calculate the parameters of the distribution of the average of n correlated samples during the offline phase. In a previous work [29,31], we have shown that α can be approximated using the autocorrelation coefficient with lag 1. The variance of the distribution can be calculated using Equation 5. These distribution parameters (μ, σ, and α) are then stored in the radio map.

3.4.3 Correlation handler

During the online phase, the correlation handler module averages the value of n consecutive samples from an access point and passes this information to the discrete-space estimator, which uses the distributions stored in the radio map (taking correlation into account using the information in Section 3.4.1) to estimate the user location.

3.5 Continuous Space Estimator

The discrete-space estimator returns a single location from the set of locations in the radio map. To increase the system accuracy, the *Horus* system uses two techniques to obtain a location estimate in the continuous space.

3.5.1 Technique 1: Center of Mass of the Top Candidate Locations

This technique is based on treating each location in the radio map as an object in the physical space whose weight is equal to the normalized probability[1] assigned by the discrete-space estimator. We then obtain the center of mass of the N objects with the largest mass, where N is a parameter to the system, $1 \leq N \leq ||\mathbf{X}||$.

More formally, let $p(x)$ be the probability of a location $x \in \mathbb{X}$, i.e., the radio map, and let $\bar{\mathbb{X}}$ be the list of locations in the radio map *ordered* in a descending order according to the normalized probability (the location with lower ID comes first for locations with equal probability). The center of mass technique estimates the current location x as:

$$x = \frac{\sum_{i=1}^{N} p(i)\bar{\mathbb{X}}(i)}{\sum_{i=1}^{N} p(i)} \quad (6)$$

where $\bar{\mathbb{X}}(i)$ is the i^{th} element of $\bar{\mathbb{X}}$

Note that the estimated location x need not be one of the radio map locations.

3.5.2 Technique 2: Time-Averaging in the Physical Space

The second technique uses a time-average window to smooth the resulting location estimate. The technique obtains the location estimate by averaging the last W locations estimates obtained by either the discrete-space estimator or the continuous-space estimator discussed in the previous section.

More formally, given a stream of location estimates $x_1, x_2, ..., x_t$, the technique estimates the current location $\bar{x_t}$ at time t as:

$$\bar{x_t} = \frac{1}{\min(W,t)} \sum_{t-\min(W,t)+1}^{t} x_i \quad (7)$$

We compare the two techniques in Section 4.

3.6 Small-Scale Compensator

Dealing with small-scale variations (Figure 5) is challenging. Since the selected radio map locations are typically placed more than a meter apart, to limit the radio map size, the radio map does not capture small-scale variations. This contributes significantly to the estimation errors in the current systems. The *Horus* system uses the *Perturbation* technique to handle the small-scale variations. The technique is based on two sub-functions: detecting small-scale variations and compensating for small-scale variations.

3.6.1 Detecting small-scale variations

In order to detect small-scale variations, the *Horus* system uses the heuristic that users' location cannot change faster than their movement rate. The system calculates the estimated location using the standard radio map and the inference algorithm, then calculates the distance between the estimated location and the previous user location. If this distance is above a threshold, based on the user movement rate and estimation frequency, the system decides that there are small-scale variations affecting the signal strength.

3.6.2 Compensating for small-scale variations

To compensate for these small-scale variations, the system perturbs the received vector entries, re-estimates the location, and chooses the nearest location to the previous user location as the final location estimate. For example, if one sample includes a signal-strength observation from each of k access points $(s_1, s_2, \ldots, s_k)$, the system tries all 3^k combinations in which each of the k observations i is replaced by one of three values, s_i, $s_i(1+d)$, or $s_i(1-d)$; we explore the parameter d in Section 4.2.4. An enhancement of this approach is to perturb a subset of the access points. The effect of the number of access points to perturb and the value of d on accuracy is described in Section 4.

3.7 Clustering Module

This section describes the *Incremental Triangulation* (IT) clustering technique used by the *Horus* system to reduce the computational requirements of the location determination algorithm. We define a *cluster* as a set of locations sharing a common set of access points. We call this common set of access points the *cluster key*. The problem can be stated as: Given a location x, we want to determine the cluster to which x belongs. The noisy characteristics of the wireless channel described in Section 2 make clustering a challenging problem because the number of access points covering a location varies with time.

The *IT* approach is based on the idea that each access point defines a subset of the radio map locations that are covered by this access point. These locations can be viewed as a cluster of locations whose key is the access point covering the locations in this cluster. If during the location determination phase we use the access points incrementally, one after the other, then starting with the first access point, we restrict our search space to the locations covered by this access point. The second access point chooses only the locations in the range of the first

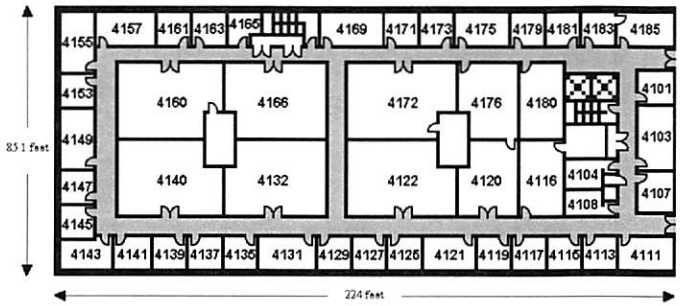


Figure 7: Floor plan for the first testbed. Readings were collected in the corridors and inside the rooms.

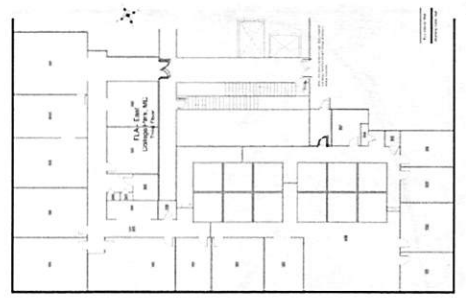


Figure 8: Floor plan of the office space where the second experiment was conducted. Readings were collected in the corridors and inside the rooms.

access point and covered by the second access point and so on, leading to a multi-level clustering process.

Notice that no preprocessing is required in the offline training phase. During the online phase, a location x belongs to a cluster whose key is access point a if there is information about access point a at location x in the radio map.

The algorithm works as follows. Given a sequence of observations from each access point, we start by sorting the access points in descending order according to the average received signal strength. For the first access point, the one with the strongest average signal strength, we calculate the probability of each location in the radio map set given the observation sequence from this access point alone. This gives us a set of candidate locations (locations that have non-zero probability). If the probability of the most probable location is "significantly" higher (according to a threshold) than the probability of the second most probable location, we return that most probable location as our location estimate, after consulting only one access point. If this is not the case, we go to the next access point in the sorted access point list. For this access point, we repeat the same process again, but only for the set of candidate locations obtained from the first access point. We study the performance of the *IT* approach in Section 4.

4 Experimental Evaluation

In this section we start by showing the effect of each module independently on the the accuracy of the basic algorithm. We then show the effect of using all the components together on the performance of the *Horus* system.

4.1 Experimental Testbed

We performed our experiment in two different testbeds.

4.1.1 Testbed 1

We performed our first experiment in the south wing of the fourth floor of the A. V. Williams building in the University of Maryland at College Park. The layout of the floor is shown in Figure 7. The wing has a dimension of 68.2 meters by 25.9 meters. The technique was tested in the University of Maryland wireless network using Cisco access points. 21 access points cover the multi-story wing and were involved in testing.

The radio map has 110 locations along the corridors and 62 locations inside the rooms. On the average, each location is covered by 6 access points. The *Horus* system was running under the Windows XP professional operating system.

4.1.2 Testbed 2

We performed the second experiment in another office space (Figure 8). The area of the experiment site is approximately 11.8 meters by 35.9 meters covering corridors, cubicles, and rooms. Five LinkSys access points and one Cisco access point cover the test area.

We have a total of 110 locations in the radio map. On the average, each location is covered by 4 access points. The *Horus* system was running under the *Linux* (kernel 2.5.7) operating system.

4.1.3 Data collection

The radio map locations were marked on the floor before the experiment and the user clicked on the map to point the location of the radio map points. We collected 100 samples, spaced 300 ms apart, at each radio map location. We expect an error of about 15-20 cm due to the inaccuracies in clicking the map.

The training data was placed 1.52 meters (5 feet) apart for the first testbed and 2.13 meters apart for the second testbed (7 feet).

For each testbed, we selected 100 test locations to random cover the entire test area (none of them coincide with a training point). For both testbeds, the test set was

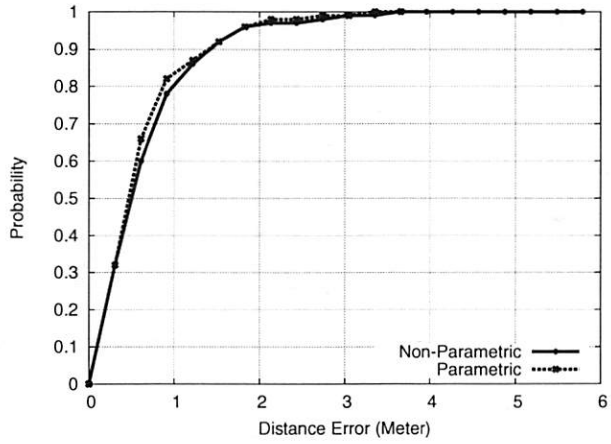


Figure 9: Performance of the basic algorithm of the *Horus* system for the first testbed.

collected by different persons on different days and times of day than the training set. This difference presents a realistic testbed and should, if anything, decrease the measured accuracy of our approach because it lessens the likelihood that the test data is a close match to the training data.

4.2 Results

We show the effect of each module independently on the performance of the discrete-space estimator and present the overall system performance in Section 4.3.

4.2.1 Discrete-space estimator (Basic algorithm)

Figure 9 shows the performance of the basic algorithms of the *Horus* system for the first testbed. The system can achieve an accuracy of 1.4 meters 90% of the time. The performance of the parametric and non-parametric methods is comparable with a slight advantage for the parametric method. Using a parametric distribution to estimate the signal-strength distribution smooths the distribution shape to account for missing signal strength values in the training phase (due to the finite training time). This smoothing avoids obtaining a zero probability for any signal strength value that was not obtained in the training phase and hence enhances the accuracy.

Table 1 shows the summary of the results for the two testbeds. Details for the second testbed can be found in [28].

4.2.2 Correlation handler

Figure 10 shows the performance of the *Horus* system when taking the correlation into account and without taking the correlation into account for the first testbed. We estimated the value of α to be 0.9. The figures show that under the independence assumption, as the number of averaged samples increases, the performance degrades. The minimum value at $n = 2$ can be explained by noting that there are two opposing factors affecting the system accuracy:

1. as the number of averaged samples n increases, the accuracy of the system should increase.

2. as n increases, the estimation of the distribution of the average of the n samples becomes worse due to the wrong independence assumption.

At low values of n ($n = 1, 2$) the first factor is the dominating factor and hence the accuracy increases. Starting from $n = 3$, the effect of the bad estimation of the distribution becomes the dominating factor and accuracy degrades.

Using the modified technique, the system average accuracy is enhanced by more than 19% using five signal-strength samples.

Table 1: Summary of the percentage enhancement of different components on the basic algorithm

Technique	Testbed 1	Testbed 2
Correlation Handling	19%	11%
Center of Mass	13%	6%
Time Averaging	24%	15%
Small-Scale Compensator	25%	21%

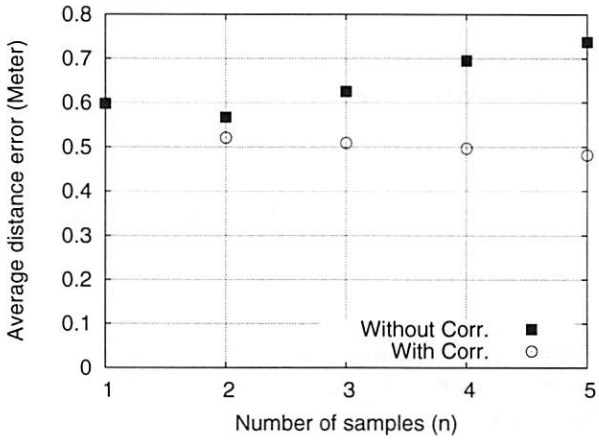


Figure 10: Average distance error with and without taking correlation into account for the first testbed.

4.2.3 Continuous space estimator

Center of Mass Technique: Figure 11 shows the effect of increasing the parameter N (number of locations to interpolate between) on the performance of the center of mass technique for the first testbed. Note that the special

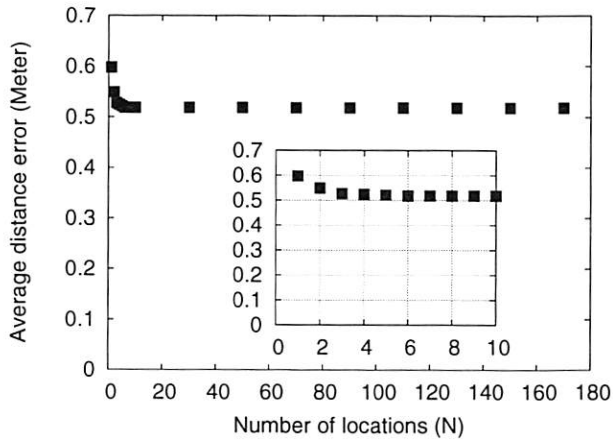


Figure 11: Average distance error using the center of mass technique for the first testbed.

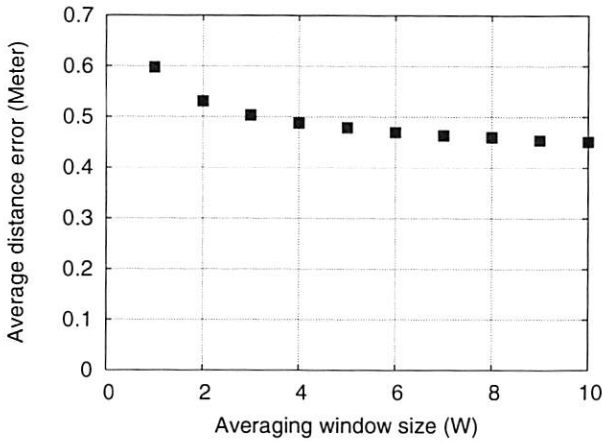


Figure 12: Average distance error using the time-averaging technique for the first testbed.

case of $N = 1$ is equivalent to the discrete-space estimator output. The figures show that the performance of the *Horus* system is enhanced by more than 13% for $N = 6$.

Time-averaging Technique: Figure 12 shows the effect of increasing the parameter W (size of the averaging window) on the performance of the time-averaging technique. The figures show that the larger the averaging window, the better the accuracy. The performance of the *Horus* system is enhanced by more than 24% for $W = 10$.

4.2.4 Small-scale compensator

For the purpose of detecting small-scale variations, we assume a maximum user speed of two meters per second.

Figure 13 shows the effect of changing the perturbation fraction (d, which is the amount by which to perturb each access point) on average error. We can see from this figure that the best value for the perturbation fraction is 0.05 for the first testbed. We use these values for the rest of this section.

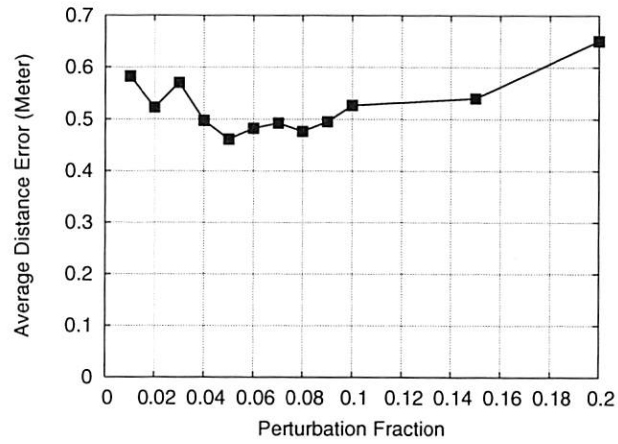


Figure 13: Effect of changing the perturbation fraction on average distance error.

Figure 14 shows the effect of increasing the number of perturbed access points on the average distance error. The access points chosen at a location are the strongest access points in the set of access points that cover that location. The figure shows that perturbation technique is not sensitive to the number of access points. This means that perturbing one access point only is sufficient to enhance the performance.

Figures 15 shows the effect of using the perturbation technique on the basic *Horus* system. The perturbation technique reduces the average distance error by more than 25% and the worst-case error is reduced by more than 30%.

4.2.5 Clustering module

Figures 16 and 17 shows the effect of the parameter $Threshold$ on the performance. For small values of the $Threshold$ parameter, the decision is taken quickly after examining a small number of access points. As the threshold value increases, more access points are consulted to reach a decision. As the number of access points consulted increases, the number of operations (multiplications) per location estimate increases and so does the accuracy.

4.3 Overall System Performance

In the previous sections, we studied the effect of each component of the *Horus* system separately on the performance. In this section, we compare the performance of the full *Horus* system, to the performance of a deterministic technique (the *Radar* system [5]) and a probabilistic

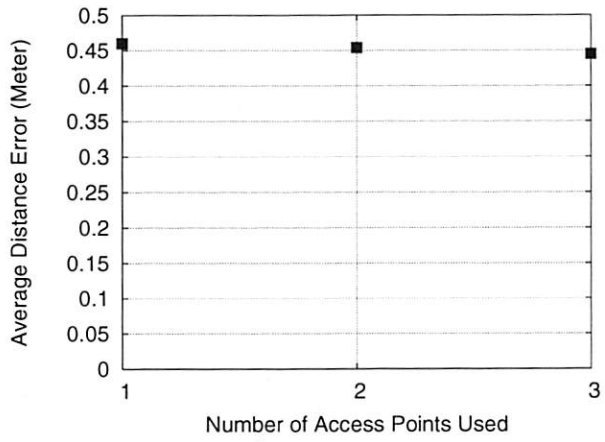


Figure 14: Effect of changing the number of perturbed access points on average distance error.

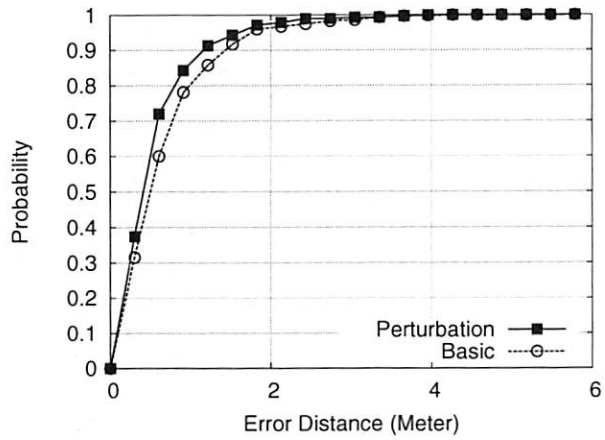


Figure 15: CDF for the distance error for the first testbed.

technique [21]. We use the parametric distribution technique. Table 2 shows the values of different parameters.

Figures 18 shows the comparison for the two testbeds (the curve for the *Radar* system is truncated). Tables 3 summarizes the results. The table shows that the average accuracy of the *Horus* system is better than the Radar system by more than 89% for the first testbed and 82% for the second testbed. The worst case error is decreased by more than 93% for the first testbed and 70% for the second testbed.

Comparing the probabilistic system in [21] to the *Horus* system shows that the average error is decreased by more than 35% for the first testbed and 27% for the second testbed. The worst case error is decreased by more than 78% for the first testbed and 70% for the second testbed. These results show the effectiveness of the proposed techniques.

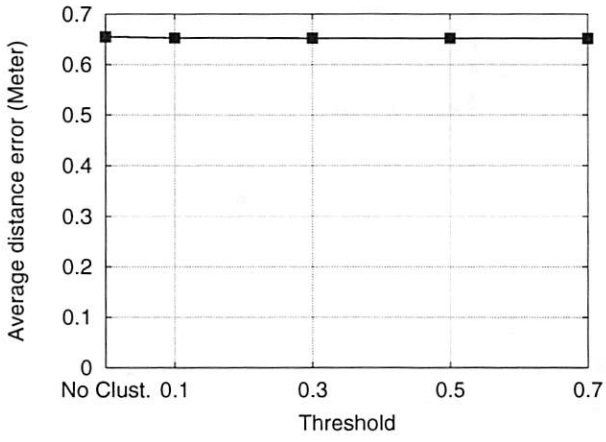


Figure 16: Effect of the parameter $Threshold$ on the average distance error for the first testbed.

Table 2: Estimation parameters for the two testbeds

Parameter	**Test. 1**	**Test. 2**
Correlation Degree (α)	0.9	0.7
Num. of avg. samples (n)	10	10
Num. of loc. used in interp. (N)	6	6
Averaging window (W)	10	10
Threshold	0.1	0.1

The performance of the three systems is better in the first testbed than the second testbed as the first testbed has a higher density of APs per location and the calibration points were closer for the first testbed.

Moreover, the *Horus* system leads to more than an order of magnitude savings in the number of multiplications required per location estimate compared to the other systems (250 multiplications for Horus compared to 2708 for the other two systems).

We also applied the enhancement discussed in this paper (without correlation handling) to the Radar system. We summarize the results is Table 3. These results show the effectiveness of the techniques proposed in the paper and that these techniques are general and can be applied to other WLAN location determination systems to enhance their accuracy.

5 Discussion

In this section, we highlight some of our experience with the *Horus* system.

Table 3: Comparison of the *Horus* system and other systems (error in centimeters)

	Testbed 1					Testbed 2				
	Median	**Avg**	**Stdev**	**90%**	**Max**	**Median**	**Avg**	**Stdev**	**90%**	**Max**
Horus	39	42	28	86	121	51	64	53	132	289
System [21]	48	65	63	143	578	72	86	77	181	991
Radar	296	400	326	853	1757	341	361	184	611	967
Radar with Horus tech.	161	193	107	302	423	142	195	106	332	483

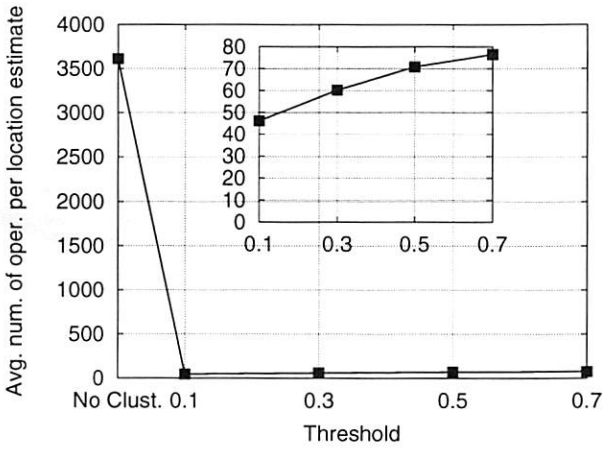


Figure 17: Effect of the parameter $Threshold$ on the average number of operations per location estimate for the first testbed. The sub-figure shows the same curve for $Threshold = [0.1, 0.7]$.

5.1 Parametric vs Non-Parametric Distributions

The *Horus* system can model the signal strength distributions received from the access points using parametric or non-parametric distributions. The main advantage of the non-parametric technique is the efficiency of calculating the location estimate, while the parametric technique reduces the radio map size and smooths the distribution shape which leads to a slight computational advantage of the parametric technique over the non-parametric technique.

5.2 Location Estimation Latency

The correlation handling and the continuous space estimator modules each use more than one sample to increase the accuracy of the system. However, a side effect of this increased accuracy is that the latency of calculating the location estimate increases. In general, we have a tradeoff between the accuracy required and the latency of the location estimate. The higher the required accuracy, the higher the number of samples required and the higher the latency to obtain the location estimate. This decision is dependent on the application in use.

Latency can be reduced by presenting the location estimate incrementally using one sample at a time. The system need not to wait until it acquires the n samples all at once. Instead, it can give a more accurate estimate of the location as more samples become available by reporting the estimated location given the partial samples it has. Other factors that affect the choice of the value of n are the user mobility rate and the sampling rate. The higher the user mobility rate or the sampling rate, the lower the value of n.

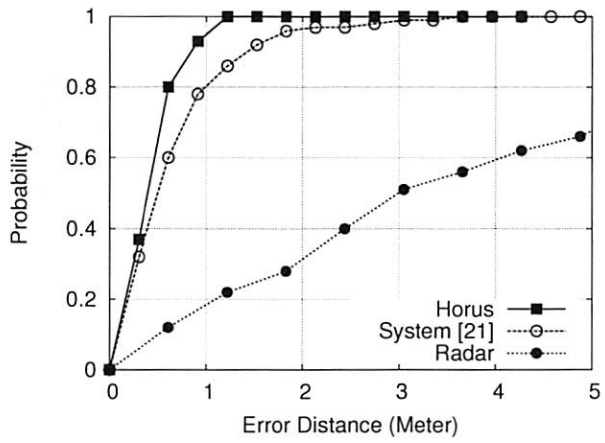
5.3 User Profile

A common assumption of WLAN location determination systems is that the user position follows a uniform distribution over the set of possible locations. Our analysis and experimentation [32] show that knowing the probability distribution of the user position can reduce the number of access points required to obtain a given accuracy. However, with a high density of access points, the performance of the *Horus* system is consistent under different probability distributions for the user position, i.e., the effect of the user profile is not significant with a high density of access points. Systems can use this fact to reduce the energy consumed in the location determination algorithm by not using the user profile in the estimation process.

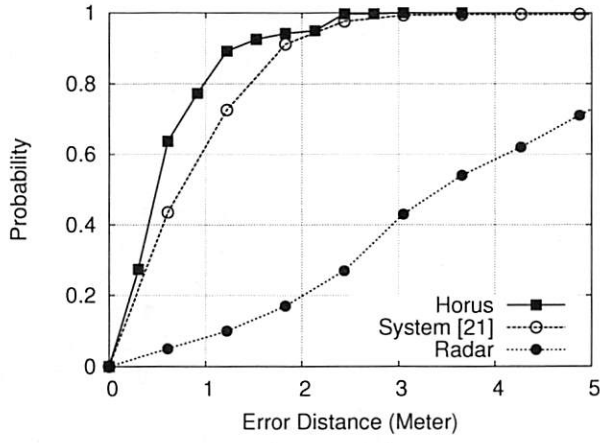
5.4 Effect of Different Hardware

One of the hardware related questions is whether different hardware from different manufacturers are compatible. That is, how does using different APs, mobile devices, or wireless cards affect the accuracy?

Our experience with the *Horus* system shows that the Laptop or PDA used for the calibration has no effect on the accuracy if a different device is used in tracking. APs from different manufacturers can be used without affecting the accuracy since the radio map captures the signature of the AP at each location (note that the second testbed uses mixed types of APs). The 802.11h specifications, however, require APs to have transmission power control (TPC) and dynamic frequency selection (DFS). This presents an open research direction for the current WLAN location determination systems as they assume that the AP transmission power does not change over



(a) First testbed



(b) Second testbed

Figure 18: CDF of the performance of the *Horus* system and the *Radar* and the probabilistic system.

time.

The main factor that may affect the accuracy when changing hardware is the wireless card. Our experience shows that cards from the same manufacture are interchangeable. The good news is that a linear mapping exists between different NICs [13]. Unfortunately, some of the cards in the market are so noisy [27] that with this linear mapping the obtained radio map is not representative of the environment. We found that Orinoco cards and Cisco cards are stable, in terms of signal-strength measurements.

5.5 Operating System Interface

We implemented the *Horus* system under both Linux and Windows. The main functionality we require from the OS is support for issuing scan requests and returning the results. Under the Linux OS, the wireless extensions [1,2] give us a common interface to query different drivers that support that interface. Under the Windows OS, NDIS allowed us to perform the same functions.

Our experience with both systems shows that drivers under Linux conform to the Wireless Extensions APIs better than Windows Drivers do with the NDIS. For example, under the Windows, some cards, like the Cisco card, respond to scans with low frequency (every 2-3 seconds) and return only one AP. We hope that future versions of the driver will have better support for the NDIS interface. Moreover for both systems, better active scanning techniques needs to be developed to reduce the scanning overhead.

6 Related Work

Many systems over the years have tackled the problem of determining and tracking the user position. Examples include GPS [11], wide-area cellular-based systems [24], infrared-based systems [4, 26], ultrasonic-based systems [19], various computer vision systems [16], and physical contact systems [18].

Compared with these systems, WLAN location determination systems are software based (do not require specialized hardware) and may provide more ubiquitous coverage. This feature adds to the value of the wireless data network.

The Daedalus project [14] developed a system for coarse-grained user location. A mobile host estimates its location to be the same as the base station to which it is attached. Therefore, the accuracy of the system is limited by the access point density.

The RADAR system [5] uses the RF signal strength as an indication of the distance between the transmitter and receiver. During an offline phase, the system builds a radio map for the RF signal strength from a fixed number of receivers. During normal operation, the RF signal strength of the mobile client is measured by a set of fixed receivers and is sent to a central controller. The central controller uses a K-nearest approach to determine the location from the radio map that best fits the collected signal strength information.

The Aura system proposed in [22] uses two techniques: pattern matching (PM) and triangulation, mapping and interpolation (TMI). The PM approach is very similar to the RADAR approach. In the TMI technique, the physical position of all the access points in the area needs to be known and a function is required to map signal strength onto distances. They generate a set of training points at each trained position. The interpolation of the training data allows the algorithm to use less training data than the PM approach. During the online phase, they use the approximate function they got from the training data to generate contours and they calculate the intersection between different contours yielding the signal space position of the user. The nearest set of mappings from

the signal-space to the physical space is found by applying a weighted average, based on proximity, to the signal space position.

The Nibble location system, from UCLA, uses a Bayesian network to infer a user location [8]. Their Bayesian network model include nodes for location, noise, and access points (sensors). The signal to noise ratio observed from an access point at a given location is taken as an indication of that location. The system also quantizes the SNR into four levels: high, medium, low, and none. The system stores the joint distribution between all the random variables of the system.

Another system, [21], uses Bayesian inversion to return the location that maximizes the probability of the received signal strength vector. The system stores the signal strength histograms in the radio map and uses them in the online phase to estimate the user location. Yet, another system, [17], applies the same technique to the robotics domain.

The *Horus* system is unique in defining the possible causes of variations in the received signal strength vector and devising techniques to overcome them, namely providing the correlation modeler, correlation handler, continuous space estimator, and small-space compensator modules. Moreover, it reduces the computational requirements of the location determination algorithm by applying location-clustering techniques. This allows the *Horus system* to achieve its goals of high accuracy and low energy consumption.

7 Conclusions

In this paper, we presented the design of the *Horus* system: a WLAN-based location determination system. We approached the problem by identifying the various causes of variations in a wireless channel and developed techniques to overcome them. We also showed the various components of the system and how they interact.

The *Horus* system models the signal strength distributions received from access points using parametric and non-parametric distributions. By exploiting the distributions, the *Horus* system reduces the effect of temporal variations.

We showed that the correlation of the samples from the same access point can be as high as 0.9. Experiments showed that under the independence assumption, as the number of averaged samples increases, the performance degrades. Therefore, we introduced the correlation modeler and handling modules that use an autoregressive model for handling the correlation between samples from the same access point. Using the modified technique, the system average accuracy is enhanced by more than 19% for the first testbed and 11% for the second testbed.

The *Horus* system uses the *Perturbation* technique for handling small-scale variations. The perturbation technique enhances the average distance error by more than 25% for the first testbed and more than 21% for the second testbed. Moreover, the worst-case error is reduced by more than 30% for the two testbeds.

The basic *Horus* technique chooses the estimated location from the discrete set of radio map locations. We described two techniques for allowing continuous-space estimation: the *Center of Mass* technique and the *Time-Averaging* technique. Using the *Center of Mass* technique, the accuracy of the *Horus* system was increased by more than 13% for the first testbed and by more than 6% for the second testbed compared to the basic technique. The *Time-Averaging* technique enhances the performance of the *Horus* system by more than 24% for the first testbed and more than 15% for the second testbed. The two techniques are independent and can be applied together.

We also compared the performance of the *Horus* system to the performance of the *Radar* system. We showed that the average accuracy of the *Horus* system is better than the Radar system by more than 89% for the first testbed and 82% for the second testbed. The worst case error is decreased by more than 93% for the first testbed and 70% for the second testbed. Comparing the probabilistic system in [21] to the *Horus* system shows that the average error is decreased by more than 35% for the first testbed and 27% for the second testbed. The worst case error is decreased by more than 78% for the first testbed and 70% for the second testbed. These results show the effectiveness of the proposed techniques. In terms of computational requirements, the *Horus* system is more efficient by more than an order of magnitude.

The proposed modules are all applicable to any of the current WLAN location determination systems. We show the result of applying the techniques of the *Horus* system to the *Radar* system. The results show that the average distance error is reduced by more than 58% for the first testbed and by more than 54% for the second testbed. The worst case error is decreased by more than 76% for the first testbed and by more than 48% for the second testbed.

As part of our ongoing work we are experimenting with different clustering techniques. Automating the radio-map generation process is a possible research area. The *Horus* system provides an API for location-aware applications and services. We are looking at designing and developing applications and services over the *Horus* system. A possible future extension is to dynamically change the system parameters based on the environment, such as changing the averaging window size as the user speed changes or using a time-dependent radio map. We are also working on the theoretical analysis of different components of the system.

Our experience with the *Horus* system showed that it has achieved its goals of:

- High accuracy: through a probabilistic location determination technique, using a continuous-space estimator, handling the high correlation between samples from the same access point, and the perturbation technique to handle small-scale variations.

- Low computational requirements: through the use of clustering techniques.

The design of Horus also allows it to achieve scalability to large coverage areas, through the use of clustering techniques, and to large number of users, through the distributed implementation on the mobile devices and due to the low energy requirements of the algorithms.

Moreover, the techniques presented in this paper may be applicable to other RF-technologies such as 802.11a, 802.11g, HiperLAN, and BlueTooth.

Acknowledgments

This work was supported in part by the Maryland Information and Network Dynamics (MIND) Laboratory, its founding partner Fujitsu Laboratories of America, and by the Department of Defense through a University of Maryland Institute for Advanced Computer Studies (UMIACS) contract.

Availability

The MAPI API and the Linux device drivers are available for download at [1].

References

[1] http://www.cs.umd.edu/users/moustafa/Downloads.htm.
[2] http://www.hpl.hp.com/personal/Jean_Tourrilhes/.
[3] http://www.orinocowireless.com.
[4] AZUMA, R. Tracking requirements for augmented reality. *Communications of the ACM 36*, 7 (July 1997).
[5] BAHL, P., AND PADMANABHAN, V. N. RADAR: An In-Building RF-based User Location and Tracking System. In *IEEE Infocom 2000* (March 2000), vol. 2, pp. 775–784.
[6] BAHL, P., PADMANABHAN, V. N., AND BALACHANDRAN, A. Enhancements to the RADAR User Location and Tracking System. Tech. Rep. MSR-TR-00-12, Microsoft Research, February 2000.
[7] BOX, G. E. P., JENKINS, G. M., AND REINSEL, G. C. *Time Series Analysis: Forcasting and Control*, third ed. Prentice Hall, 1994.
[8] CASTRO, P., CHIU, P., KREMENEK, T., AND MUNTZ, R. A Probabilistic Location Service for Wireless Network Environments. *Ubiquitous Computing 2001* (September 2001).
[9] CASTRO, P., AND MUNTZ, R. Managing Context for Smart Spaces. *IEEE Personal Communications* (OCTOBER 2000).
[10] CHEN, G., AND KOTZ, D. A Survey of Context-Aware Mobile Computing Research. Tech. Rep. Dartmouth Computer Science Technical Report TR2000-381, 2000.
[11] ENGE, P., AND MISRA, P. Special issue on GPS: The Global Positioning System. *Proceedings of the IEEE* (January 1999), 3–172.
[12] GWON, Y., JAIN, R., AND KAWAHARA, T. Robust Indoor Location Estimation of Stationary and Mobile Users. In *IEEE Infocom* (March 2004).
[13] HAEBERLEN, A., FLANNERY, E., LADD, A., RUDYS, A., WALLACH, D., AND KAVRAKI, L. Practical Robust Localization over Large-Scale 802.11 Wireless Networks. In *10th ACM MOBICOM* (Philadelphia, PA, September 2004).
[14] HODES, T. D., KATZ, R. H., SCHREIBER, E. S., AND ROWE, L. Composable ad hoc mobile services for universal interaction. In *3rd ACM MOBICOM* (September 1997), pp. 1–12.
[15] KRISHNAN, P., KRISHNAKUMAR, A., JU, W. H., MALLOWS, C., AND GANU, S. A System for LEASE: Location Estimation Assisted by Stationary Emitters for Indoor RF Wireless Networks. In *IEEE Infocom* (March 2004).
[16] KRUMM, J., ET AL. Multi-camera multi-person tracking for Easy Living. In *3rd IEEE Int'l Workshop on Visual Surveillance* (Piscataway, NJ, 2000), pp. 3–10.
[17] LADD, A. M., BEKRIS, K., RUDYS, A., MARCEAU, G., KAVRAKI, L. E., AND WALLACH, D. S. Robotics-Based Location Sensing using Wireless Ethernet. In *8th ACM MOBICOM* (Atlanta, GA, September 2002).
[18] ORR, R. J., AND ABOWD, G. D. The Smart Floor: A Mechanism for Natural User Identification and Tracking. In *Conference on Human Factors in Computing Systems (CHI 2000)* (The Hague, Netherlands, April 2000), pp. 1–6.
[19] PRIYANTHA, N. B., CHAKRABORTY, A., AND BALAKRISHNAN, H. The Cricket Location-Support system. In *6th ACM MOBICOM* (Boston, MA, August 2000).
[20] ROOS, T., MYLLYMAKI, P., AND TIRRI, H. A Statistical Modeling Approach to Location Estimation. *IEEE Transactions on Mobile Computing 1*, 1 (January-March 2002), 59–69.
[21] ROOS, T., MYLLYMAKI, P., TIRRI, H., MISIKANGAS, P., AND SIEVANEN, J. A Probabilistic Approach to WLAN User Location Estimation. *International Journal of Wireless Information Networks 9*, 3 (July 2002).
[22] SMAILAGIC, A., SIEWIOREK, D. P., ANHALT, J., KOGAN, D., AND WANG, Y. Location Sensing and Privacy in a Context Aware Computing Environment. *Pervasive Computing* (2001).
[23] STALLINGS, W. *Wireless Communications and Networks*, first ed. Prentice Hall, 2002.
[24] TEKINAY, S. Special issue on Wireless Geolocation Systems and Services. *IEEE Communications Magazine* (April 1998).
[25] THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC. IEEE Standard 802.11 - Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications.
[26] WANT, R., HOPPER, A., FALCO, V., AND GIBBONS, J. The Active Badge Location System. *ACM Transactions on Information Systems 10*, 1 (January 1992), 91–102.
[27] YEO, J., BANERJEE, S., AND AGRAWALA, A. Measuring traffic on the wireless medium: experience and pitfalls. In *Technical Report, CS-TR 4421, Department of Computer Science, University of Maryland, College Park* (Dec. 2002).
[28] YOUSSEF, M. *Horus: A WLAN-Based Indoor Location Determination System*. PhD thesis, University of Maryland at College Park, May 2004. Submitted for SigMobile Dissertation Page.
[29] YOUSSEF, M., ABDALLAH, M., AND AGRAWALA, A. Multivariate Analysis for Probabilistic WLAN Location Determination Systems. In *The Second Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services* (July 2005).
[30] YOUSSEF, M., AND AGRAWALA, A. Small-Scale Compensation for WLAN Location Determination Systems. In *IEEE WCNC 2003* (March 2003).
[31] YOUSSEF, M., AND AGRAWALA, A. Handling Samples Correlation in the Horus System. In *IEEE Infocom* (March 2004).
[32] YOUSSEF, M., AND AGRAWALA, A. On the Optimality of WLAN Location Determination Systems. In *Communication Networks and Distributed Systems Modeling and Simulation Conference* (January 2004).
[33] YOUSSEF, M., AGRAWALA, A., AND SHANKAR, A. U. WLAN Location Determination via Clustering and Probability Distributions. In *IEEE PerCom 2003* (March 2003).
[34] YOUSSEF, M., AGRAWALA, A., SHANKAR, A. U., AND NOH, S. H. A Probabilistic Clustering-Based Indoor Location Determination System. Tech. Rep. UMIACS-TR 2002-30 and CS-TR 4350, University of Maryland, College Park, March 2002. http://www.cs.umd.edu/Library/TRs/.

Notes

[1]The normalization is used to ensure that the sum of the probabilities of all locations equals one.

Deploying and Evaluating a Location-Aware System

R. K. Harle
Computer Laboratory
University of Cambridge, UK
Robert.Harle@cl.cam.ac.uk

A. Hopper
Computer Laboratory
University of Cambridge, UK
Andy.Hopper@cl.cam.ac.uk

Abstract

Location-aware systems are typically deployed on a small scale and evaluated technically, in terms of absolute errors. In this paper, the authors present their experience of deploying an indoor location system (the Bat system) over a larger area and running it for a period exceeding two years.

A number of technical considerations are highlighted: a need to consider aesthetics throughout deployment, the disadvantages of specialising sensors for location only, the need for autonomous maintenance of the computational world model, the dangers in coinciding physical and symbolic boundaries, the need to design for space usage rather than space and the need to incorporate feedback mechanisms and power management. An evaluation of long term user experiences is presented, derived from a survey, logged usage data, and empirical observations. Statistically, it is found that 35% wear their Bat daily, 35% characterise their Bat as useful, privacy concerns are rare for almost 90% of users, and users cite the introduction of more applications and the adoption of the system by other users as their chief incentives to be tracked.

Thia paper aims to highlight the need to evaluate large-scale deployments of such systems both technically and through user studies.

1 Introduction

Location-aware computing is an emerging field where the location of people and objects can be used by machines to derive contextual information with which to enhance and assist users in all aspects of their lives. Indoor environments are of particular interest, potentially requiring high precision location information (of the order of centimetres) to infer useful contextual clues from the typically dense distribution of small objects such as computers, telephones, and chairs. Many indoor location systems have been implemented in attempts to achieve such high precision. Over time the field has seen positioning solutions that utilise infrared (room-based accuracy [6, 12, 13]), ultrasound (centimetre accuracy [7, 10, 14]), visible light (10-100cm accuracy [8, 11]), wireless LAN (metre accuracy [1, 2, 15]) and Ultra Wideband (UWB, approximately 15cm accuracy).

Historically, deployments of high precision indoor location systems have not involved large coverage areas due to the cost of instrumenting the environment. The most common approach is to instrument a single room, which then acts as a testbed for the location technology. However, this approach fails to represent a deployment from which usage characteristics can be reliably derived. Usage of such a room is likely to be sporadic and not representative of how users would use the space normally. In one attempt to address this, AT&T Research Cambridge and the University of Cambridge jointly developed the Bat system and deployed over a greater area.

2 The Bat System

The Bat system is an ultrasonic location system that makes use of a small, powered tag known as a 'Bat' (Figure 1). Bats are encased in sturdy plastic, with approximate dimensions of $8.0 \times 4.0 \times 1.5$cm and a weight of 50g. They can be worn on a necklace or attached to a belt clip, according to preference. Bats emit 40kHz ultrasonic pulses on command from a 433MHz radio channel. These pulses are received by a matrix of receivers in the ceiling, each accurately surveyed for position. Each receiver records the time elapsed between pulse emission and reception, thereby allowing determination of each Bat-receiver separation and hence calculation of the Bat position using a multilateration algorithm. Positional accuracy has been previously determined as within 3cm 95% of the time.

A number of applications are presently associated with the Bat system, the most salient of which are as follows:

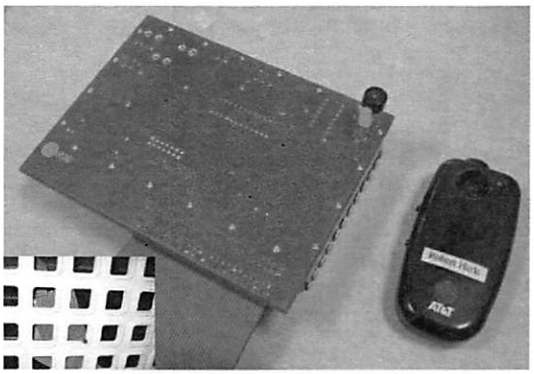


Figure 1: A ceiling receiver (left) and a Bat (right). Inset: An installed receiver

Map A graphical map of the coverage area designed for desktop machines.

Broadband Phone Map A privacy-oriented version of the map, designed to run on a networked broadband phone installed on every desk (Figure 2).

Teleporting/hotdesking Users can 'teleport' any of a number of computer desktops to their nearest machine for truly mobile working.

Access Control A display with a touch screen interface is mounted at the entrance to the laboratory. Visitors are presented with a list of tracked users who they can alert to their presence and then be directed to the current location of their host.

Pointing device Bats can be used as a three-dimensional pointing device, analogous to a desktop mouse.

Scanner The Bat system provides a convenient and intuitive interface to a number of peripherals, including a scanner.

Gaming A few simple location-based games have been developed.

User-written There is a large number of small applications that users have developed mostly for personal use. These include alerts for people returning to the office, new email notification, fresh coffee notification, diary reminders, etc. [9].

The initial deployment of the Bat system at AT&T Research Cambridge covered three floors of a building and provided positioning almost throughout the volume. In this deployment, additional applications were available, including location-based telephone call routing and location-based physical access control. Unfortunately, this deployment is no longer in place.

Figure 2: A broadband phone showing a tracking application

3 Redeploying the Bat system

Following the decommissioning of the initial deployment, the Bat system was redeployed at Cambridge in 2002. Since then the authors have accrued some valuable general lessons in deploying a pervasive location system over a large area, which are shared here. The lessons are broadly classified into system and user issues. The former deals with issues caused by installing and maintaining the system, whilst the latter considers the experiences of the day to day users.

The redeployment covers a floor area of approximately $450m^2$ and contains seventeen offices and five communal areas (Figure 3). Positioning coverage is throughout the volume, except for within a small kitchen area and the bathrooms.

The deployment was performed as the last stage of building work, after the structural components were installed, but before office furnishings and users were present. Installation, configuration and troubleshooting took approximately eight weeks, during which a number of general issues concerned with the installation and running of the system hardware were identified.

4 System Issues

Mistakes are expected in the deployment of any prototype hardware over a large volume. Here we present a summary of the lessons generated from the deployment.

4.1 Design to be aesthetically pleasing and for minimal distraction

Aesthetics are often overlooked when generating prototype hardware. However, evaluation over a medium- to large- scale deployment means that the hardware will penetrate into the working area of a wide variety of people and aesthetics are vitally important to ensure a pleasant working environment is maintained. The Bat system receivers were designed to be installed above a false

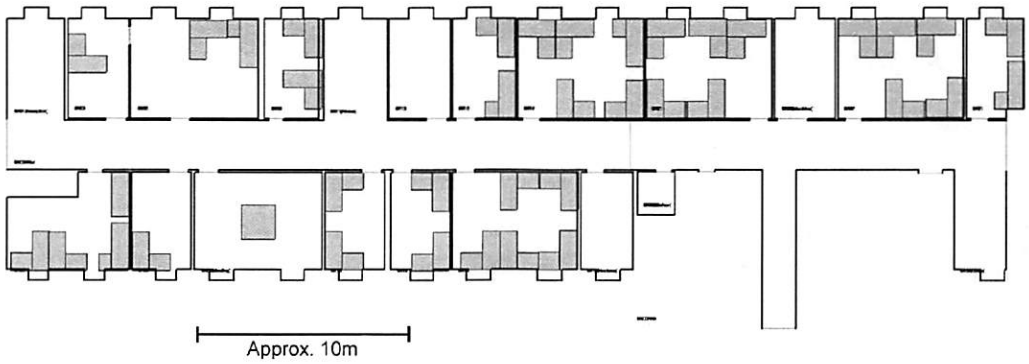


Figure 3: The deployment area of the Bat system at Cambridge

ceiling and as such hidden from view. Thus aesthetics at design time were of little concern and the hardware consists of open circuit board shielded by a metal box and interconnected by blue 50-way ribbon cable (Figure 1). These design choices proved a constraint when deploying in the laboratory: a false ceiling had to be installed at extra cost and even then it was required to have 'egg crate' ceiling tiles to prevent disruption of air flow in the building (inset, Figure 1). Hence the receivers and their interconnects are more visible than planned. Moreover, the displeasing aesthetics of the hardware has hindered subsequent attempts to redistribute the receivers in a more spatially homogeneous fashion.

4.2 Design for space usage rather than physical space

When deploying Bat receivers in the ceiling, selecting the ideal spatial distribution proved very difficult. Regular geometric patterns are not conducive to position calculation, whilst irregular distributions have negative aesthetics. In the current deployment, receiver locations were chosen to to avoid regular geometry and to give a reasonable spatial distribution for a sighting at any point in a room (Figure 4). The distributions derived from the combination of practicalities (ceiling accessibility, cable lengths) and the results of an optimisation simulation. This simulation aimed to find ceiling distributions that gave good and uniform positional coverage for the majority of the volume served. A regular three-dimensional grid of test points was defined for each room and the number of receivers set, based on room area. The simulated receivers were then allowed to move, with each individual configuration given an 'energy' state based on calculating dilution of precision metrics across the test points. To settle on a final distribution, a simulated annealing approach was used to identify good spatial arrangements.

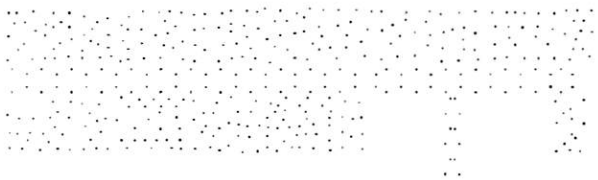
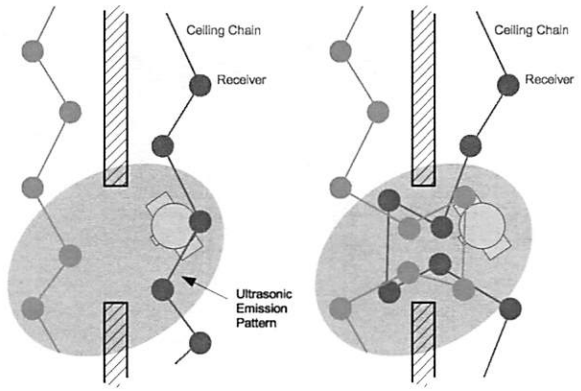


Figure 4: Receiver distribution (compare with Figure 3)

Since installation, however, it has become apparent that the distribution of sensors in the ceiling would be better deployed according to the regular *use* of the space, rather than its size and shape. For example, to achieve uniform positioning capabilities throughout a room typically required the slight degradation of positioning in one area to improve positioning elsewhere. Once office furniture was installed, it became apparent that uniform coverage was often unnecessary; users rarely use the whole of the space uniformly and the deployment distribution would be better tailored to the usage of the space. This favours the deployment of such a system *after* the space usage has been established. This complicates any deployment, but should improve performance.

4.3 Beware the coincidence of physical and symbolic boundaries

The Bat system was designed to be modular in operation, distributing the positioning calculations across multiple DSPs. Since ultrasound does not penetrate walls, a decision was made that each room would contain an independent chain of receivers and an independent DSP to handle any local multilateration calculations. Hence the system was segmented according to physical boundaries. In retrospect, however, this was a sub-optimal approach because it hinders reliable capture of room change events (since a handover from one receiver chain to another is necessary). Room changes are important location events



(a) A physical boundary means that insufficient signals are received in any one chain

(b) A spatial mixing of chains overcomes the problem

Figure 5: Chain mixing at physical boundaries

since many applications trigger events when they occur. The per-room distribution of receiver chains effectively forces a Bat to transition from one chain to another when visibility of the sensors in either chain is at its lowest. Thus we observe positioning failures near doorways, resulting in delays for room change events (Figure 5(a)).

Abstracting from this experience, we find that it is not always ideal to coincide physical boundaries with virtual boundaries (in this case th physical boundary of the chains and the boundary between two symbolic room entities). In this particular example, the issue could be addressed by the inclusion of spatial chain mixing at transition points (Figure 5(b)). Equally, a system redesign could dispense with the notion of multiple sensor chains completely, but this would require an offload of all positioning calculations to a central processor, hindering scalability.

4.4 Incorporate autonomous maintenance

Perhaps the most difficult aspect of maintaining a working location-aware system has proved to be maintenance of the computational world model: a spatial database of objects from which context is inferred from location data. A more accurate model may allow for a less accurate location system. For example, a user and computer both tracked to within 0.5m can be replaced with a user tracked to within 1m, and a computer at a known location. Thus, the creation and maintenance of an accurate world model is very important.

To keep the model synchronous with the physical world requires the precise tracking of people, furniture, devices, books, etc. Bats (and indeed, any powered tags) are not generally suitable for all classes of object — the infeasibility of regular battery changes, the added weight and cost make their deployment on truly everything prohibitive.

At present the world model used in the laboratory is hand-maintained, with changes being incorporated into the spatial database manually when reported or observed. Users of the Bat system have not felt it their responsibility to update the model when they effect any changes on the physical world, despite having software tools available to do so. Thus, maintenance is primarily achieved by manual intervention of the group members who make use of the location data. This manual approach is barely practicable on the laboratory scale, and limits the tracked objects to more bulky items that are less likely to move — the database has the capability to model arbitrary devices and objects, but the difficulties in maintaining it manually has resulted in only bulky furniture and computer hosts being modelled.. Any increase in deployment scale (without a corresponding increase in maintenance personnel) would render the approach altogether infeasible.

Over time, then, we have observed the world model to fall out of synchronisation with the real world and to require manual correction since the system does not incorporate an autonomous world model monitoring component. We have found that the loss of this synchronisation is of great significance: users become irate with the system, which does not perform as expected due to incorrect contextual inferences, and rejection of the entire location system can follow suit. Autonomous maintenance has proved to be a very difficult problem and we have actively researched ways in which it might be performed, identifying a number of starting points deriving from sighting distributions [5], and analysis of the ultrasonic signals in the system [3, 4]. Future work will incorporate different tracking technologies for different classes of object.

4.5 Design to maximise *context* gathering

In examining the signals within the Bat system it has become apparent that the system could, in principle, be used to gather further contextual information. In practice, however, its design as a positioning system has resulted in optimisations that hinder extraction of this data. The major optimisation is the choice of ceiling mounted receivers: for Bats worn at chest height, this maximises receiver visibility. The Bats have been further optimised such that ultrasonic emissions are primarily upwards. Whilst this increases power efficiency, the inefficiency of allowing a Bat to emit ultrasound more homogeneously

would benefit location-aware computing since the associated increased penetration of ultrasound into the surrounding environment allows inferences about the locality and assists in autonomous maintenance as previously mentioned.

Moreover, alongside more homogeneous spatial emissions of ultrasound from a Bat, it is useful to have a more homogeneous spatial distribution of receivers throughout a volume. It is infeasible to locate receivers in a truly homogeneous manner throughout three-dimensional space, but siting receivers at a variety of heights on the walls and outskirts of a room is possible (albeit aesthetically displeasing). The extra density may not increase positional accuracy significantly, but may allow for better signal penetration and positioning below objects (the ceiling-mounted system suffers from a loss of tracking if a tracked object lies beneath another, for example robots cannot be tracked under tables. Receiver visibility would be dramatically improved by receivers being placed at lower heights).

A distribution of receivers across the space rather than the ceiling also offers the ability to more accurately estimate the *orientation* of the user. At present, a coarse estimation is made by examining ultrasonic power distributions and shadowing (assuming users block the ultrasound to receivers behind them). A series of receivers placed around the vertical centre of the room would offer much better orientation estimation since it would allow for measurement of power distributions in the horizontal plane including the Bat. Orientation is a useful contextual clue and it is advisable for new location systems to offer the capability to detect it reliably.

In summary, the optimisations needed for a positioning system designed solely for positioning *may* differ from those for a positioning system designed to underpin an indoor location-aware service.

4.6 Beware false positives and false negatives

Experiences using the Bat system on a daily basis have highlighted that the location information is rarely definitive due to an unfortunate side effect of using a tag-based tracking system: false negatives and false positives.

False negatives occur when an application interprets a lack of sightings for a user as an indication of absence. For example, if a user cannot locate another through the Bat system they may assume they are not present. In fact they are faced with an ambiguity: is the user away or simply not wearing their Bat? The ambiguity can only be resolved by using traditional methods (physical sight, telephone calls, etc) and hence most users default to these initially.

False positives are most often associated with Bats being physically unassociated with the corresponding user location. So, for example, a user may leave their Bat on their desk at lunchtime and the system continues to report that they are tracked and within their office. This is misleading data that leads to confusion and annoyance.

To combat false negatives and false positives the Bat system supports the idea of a *quiet zone* — a small spatial region individual to each user within which the user's Bat will not be tracked. Users can thus leave their Bats in these zones if they do not wish to be tracked or as a way to tell the system that they are not present (although ambiguity still exists between these two states). False positives are prevented (assuming the user either wears their Bat or returns it to their quiet zone). False negatives still remain, since users can forget to take their Bat out of their quiet zone. As a by-product, Bats can be put into a sleep state when in such zones, conserving power. Jitter switches are used to rouse Bats when next picked up.

Initial experiences with quiet zones revealed that few users could be relied upon to return their Bat to their personal quiet zone despite having been the ones to locate them, making false positives an issue once more. Users often placed their Bats outside their zone since they had not chosen a zone with strong physical markers that allowed them to accurately recover the zone location.

To encourage the use of quiet zones and minimise incorrect context derivations a global quiet zone (a physically obvious communal region within which *all* Bats sleep) was introduced at the exit of our area and the system software was altered to emit a beep from any Bat when it entered the sleep state. A noticeable reduction in the number of Bats left outside quiet zones has occurred.

4.7 Include feedback capabilities

The introduction of a beep when Bats enter quiet zones is an example of the importance of including feedback in applications, a concept that has proved to be very important. The need for feedback has long been touted by the HCI domain, but it becomes even more important in location-aware computing, where traditional input/output devices are not available; a location-based system performing an action such as unlocking a door as a user approaches cannot assume there is a convenient visual display by the door to indicate the success (or failure) of the action. Whatever input device is used to enable, disable or configure location-aware applications must include some form of feedback to tell the user that input was successful. Similarly error conditions (such as the door failing to unlock) must be reported to give users an intuitive feeling of what is occurring and to allow them to retain a feeling of control. The inclusion of a simple tone-generating speaker in each Bat has been

extremely useful: so much so, in fact, that the ten pre-configured tunes in the Bat firmware are becoming insufficient to allow communication of the many feedback messages that now exist (each alert and error alone needs a different tone or sequence of tones).

4.8 Include power management

In any active tag-based system, power management is a very important issue. Current battery technology means lightweight tags may not offer the optimum lifetime. Two approaches to battery lifetime are possible:

Minimise power consumption. Reduce power consumption to maximise battery lifetime.

Fit recharging into human cycle. Many users will have a daily cycle that allows for recharging with minimum irritation. For example, Bats could have been designed to support a 12-hour charge, and to be recharged overnight or when not in use. This would make them more lightweight, but cause issues when recharging is incomplete or accidentally forgotten about.

The Bat system as implemented opts for the former approach. In doing so, it uses a high capacity 3.6V battery in the AA form-factor. Regular battery changes are unacceptable, so each Bat features extensive power saving features: jitter switches are used to determine whether the tracked object is moving significantly, and the polling rate dynamically adapted to avoid fast polling when the user is seated or stationary. Very high polling rates are reserved for use with applications that need them. This provides a battery lifetime of approximately eighteen months for a Bat in regular and normal use. Furthermore, an on-board voltage monitor on each Bat can be interrogated so Bats with a dying battery can be identified and batteries changed before a loss of service occurs. Experiences show this to be a very useful feature and for the battery lifetime to be sufficiently long not to cause irritation.

A down side to such a lengthy power lifetime has been that users treat Bats as perpetual machines, assuming a hardware fault rather than a lack of power when the Bat stops functioning properly. However, the inclusion of a power-monitoring hardware component that can be remotely interrogated has allowed for recent battery failures to be pre-empted.

5 User Issues

Over the years the deployed Bat system has been available to all members of the laboratory and many visitors. The laboratory contains a significant mix of technical users with different specialisations: some users work within location-aware computing, some within high speed networking, and yet others in radio communications. The advantage of the laboratory deployment, then, is that the user base is not solely those who developed the system (as is often the case). There is no reason for users to wear a Bat unless they choose to do so: the system is purely opt-in.

Computed locations are logged and archived each day. The large number of events that flow through the system are not logged unless a fault diagnosis is required. This limits the conclusions we can draw from the archived data, especially since false negatives and false positives are both common in the logs. We have filtered the log data to attempt to identify Bats that were stationary (minimising false positives), but we have no mechanism to determine whether a user was present on a particular day and chose not to wear their Bat. To further gain insight into the usage of the system and the feelings of its long term users, we issued a short survey to all members of the laboratory, whether they had a Bat assigned or not (approximately 30 users). Twenty four responses to the survey were received, failing to represent the entire population. The makeup of the members assigned Bats is predominantly male (in the current deployment, a total of four women have been assigned Bats), to the extent that we cannot reasonably draw conclusions about differences between sexes.

We recognise that the survey results are not guaranteed to be bias-free but we use them, in conjunction with the logs we have archived, to illustrate the characteristics that have established themselves empirically over time through informal discussions and observations. Their use is also given in the hope it will encourage more detailed evaluations of similar systems and their users.

The results of the small study have been divided into three major categories; usage, applications, and user classifications.

5.1 Usage

Since installation, all but three members of the laboratory have been assigned Bats (the others requesting not to be). The survey revealed that 77.3% considered wearing a Bat to be unnoticeable or an occasional distraction, whilst 13.6% found it annoying (Figure 6). It is difficult to design a robust powered tag that can be worn so as not to be obscured by other clothing or different human positions. For example, attachment to a belt may be more comfortable, but introduces problems when the tag is obscured by a jacket, or by a table for seated users. Wearing it at chest height (using a necklace) maintains a good visibility of ceiling receivers but does not suit some users, who find the occasional swinging motion annoy-

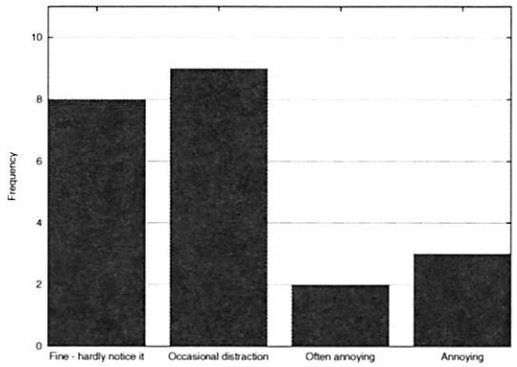


Figure 6: User-rated comfort of wearing a Bat

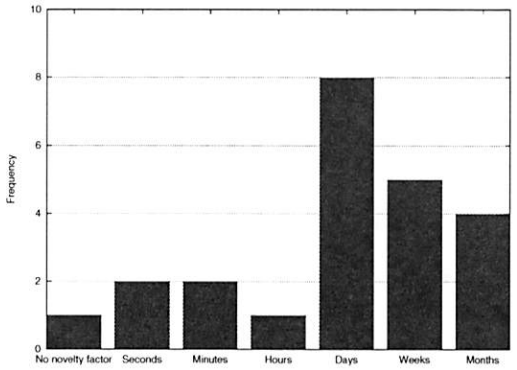


Figure 7: Duration of the novelty period when assigned a Bat

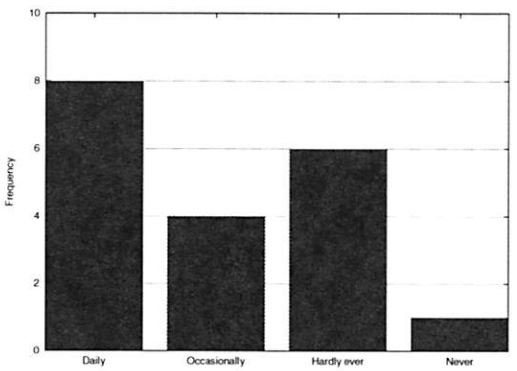


Figure 8: Frequency of wearing a Bat

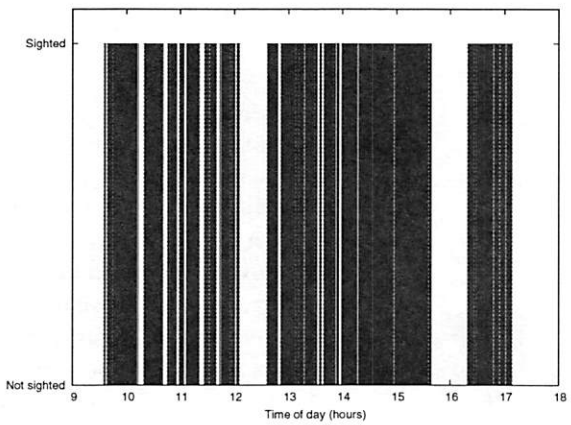
ing. One user commented that, having worn the Bat daily for some time, he now finds it more noticeable when he isn't wearing it.

Figure 9 shows the usage of a Bat for a frequent system user. The usage is typical, with the Bat being worn for the majority of the day. However, not all users use their Bat in the same way: Figure 10 shows the average number of sightings for nine (anonymised) users, captured over a 27 month period. The actual values cannot be taken as absolute since false positives may exist and an active user may accrue more sightings within a given time period than a less active colleague due to the variable polling rate. However, averaging each point over a month should minimise the impact of such concerns. It is apparent from the graph that Bat usage varied dramatically for all users over the 27 months. Of particular interest is the peak usages recorded in January 2003 — this corresponds to the full roll-out of the current system. What we observe is a novelty period during which the users wore their Bats religiously. A few months later, the average sightings per day have dropped significantly.

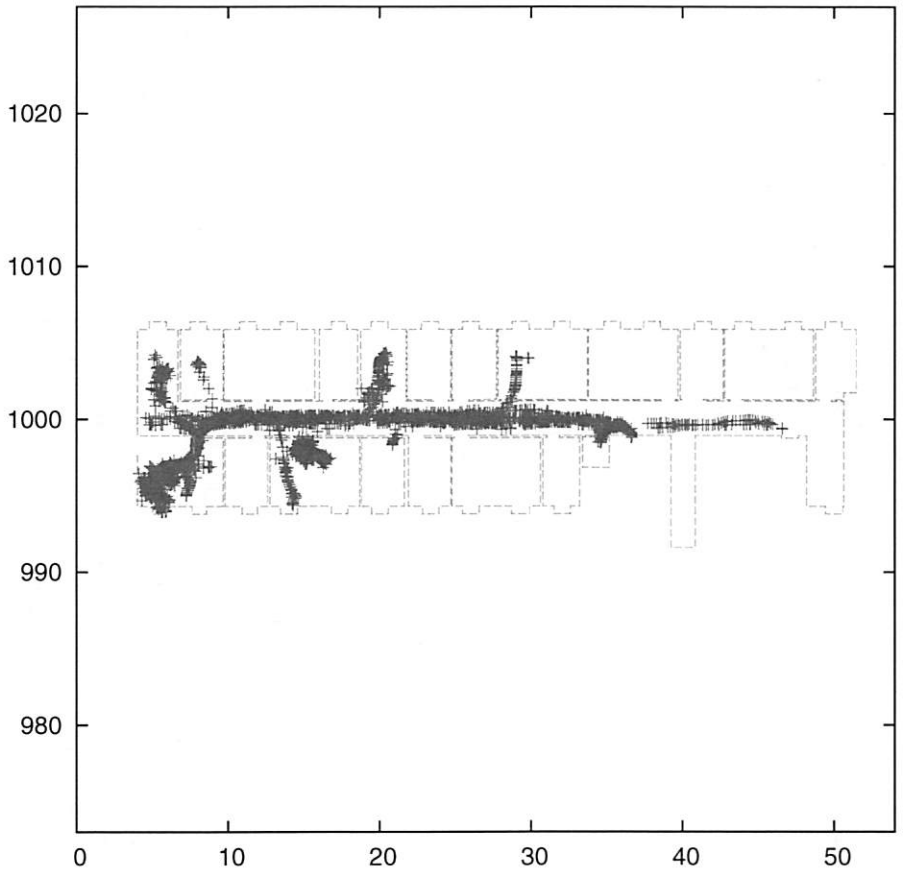
The survey revealed that users also recognise this period, but they reported a wide range of values for the duration of it (Figure 7). It seems reasonable to assume users will experience at least a few days of novelty interest. This highlights the importance of deploying a system with a wide variety of applications immediately available. User curiosity and interest is greatest during the novelty period and a readily available set of applications offers a powerful opportunity to demonstrate what location-aware computing can offer them and for them to integrate it into their routines. In the laboratory deployment, emphasis was placed on achieving a robust system, and not all applications were rolled out simultaneously (indeed, some were yet to be conceived).

Following the novelty factor usage often drops: the survey indicated only 42.1% of surveyed users wear their Bat on a daily basis and 31.5% hardly ever (Figure 8). Figure 11 shows a frequency count for the daily number of distinct users of the system over five months in 2004 (chosen to minimise the likelihood of holidays and excluding weekends). To reduce false positives, a user was included only if they were observed to move more than 1m within the day. The histogram data suggests an average of 12.2 people using their Bat daily, which is approximately 40% of the users currently assigned Bats. This supports the survey results, but the histogram further illustrates a high variability in usage,

When asked to characterise their Bat a wide variety of responses were given (Figure 12), illustrating how different users perceive its usefulness. The majority of users appear to recognise that location-aware computing can contribute favourably to their lives, but these results suggest that at present not every user of the system is able to



(a) Sighting times



(b) The sighting distribution built up over the day

Figure 9: Typical usage of a Bat over one day

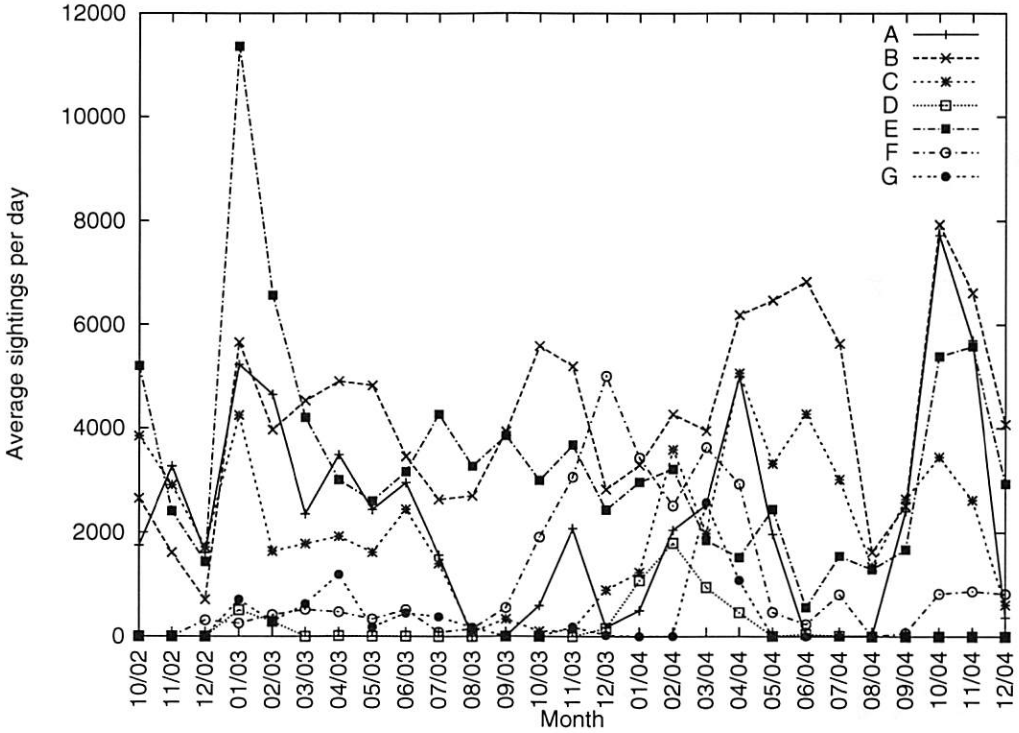


Figure 10: Logged system usage over 27 months

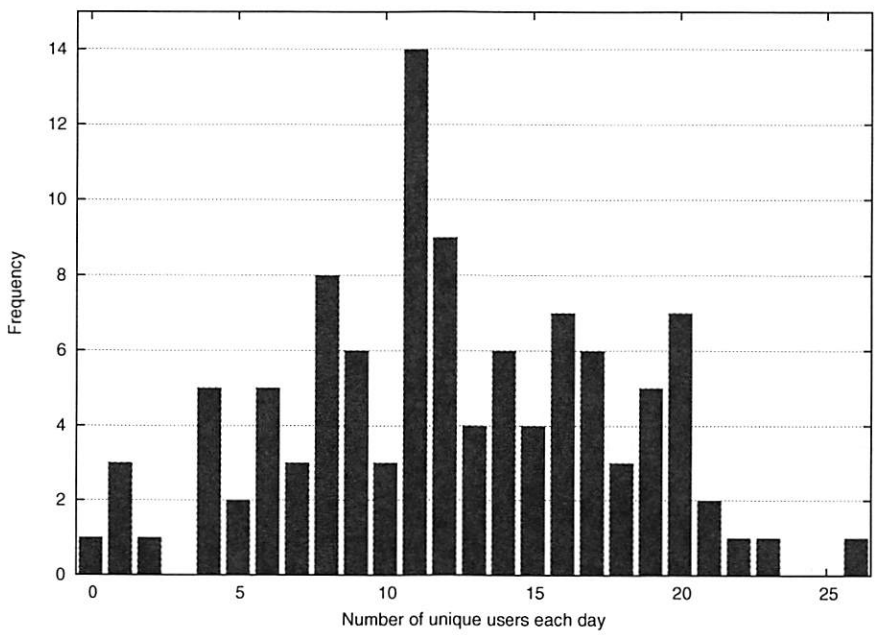


Figure 11: Unique daily users in 2004

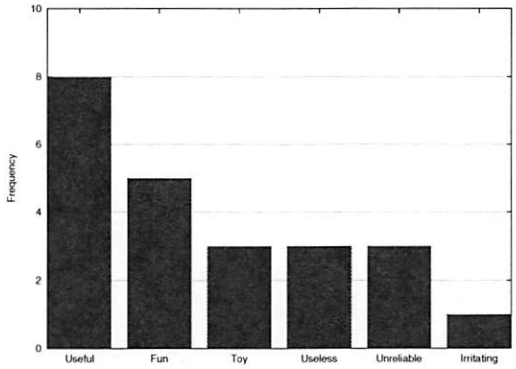


Figure 12: Characterising a Bat

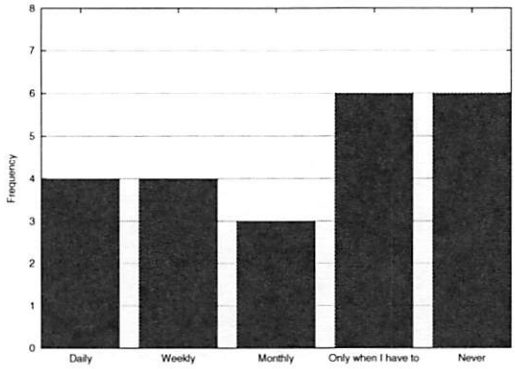


Figure 13: Frequency of location-aware application use

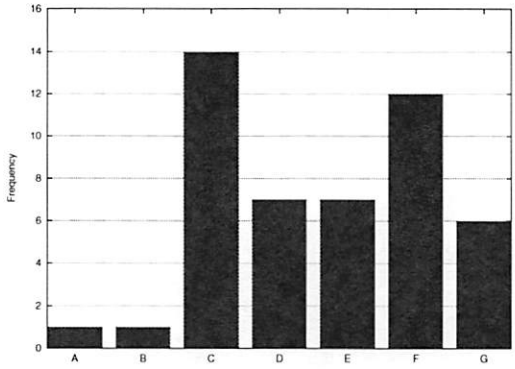


Figure 14: What would encourage the wearing of a Bat? *Key to answers: A — Nothing, B — Privacy Controls, C — More applications, D — Greater system reliability, E — More lightweight design, F — If more lab members wore one than currently do, G — A reminder to put it on*

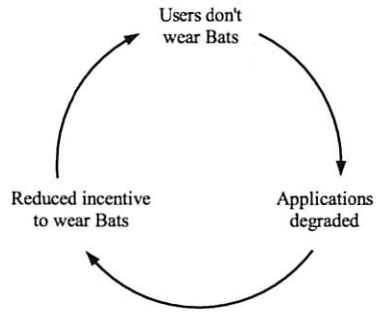


Figure 15: The pervasive application vicious cycle

identify an application they find useful.

5.2 Applications

Applications are clearly crucial in any large scale system deployment. The survey revealed a wide range in the frequency of application usage (Figure 13); a quarter only used location-aware applications when they felt they had to. Encouraging users to wear Bats and use the associated applications has been the subject of previous work [9], where specific audiences were identified and applications tailored to them. Interestingly, some users seemed also to associate a novelty period with new applications targeted at them. However, Bat usage quickly returned to its previous levels. The survey revealed that the two most popular incentives for users to increase their Bat usage were the implementation of more applications and for more users to wear their Bats (Figure 14).

Users commonly indicate that they want more applications, but can rarely identify a specific application that would be useful to them. This echoes the fact that many users are unsure about exactly what location-aware technology can offer them and are presently keen to explore.

The latter suggestion — that users won't wear Bats until others choose to — leads to a *pervasive application vicious cycle* as shown in Figure 15. Breaking such a cycle has proved difficult — the most obvious methods for doing so are the introduction of a rule that Bats must be worn (generally undesirable, giving Bats a spy-like character) or the introduction of an application perceived globally to be of use (an ongoing research area).

With regard to specific applications, the survey revealed that the most popular class of applications in use were mapping applications. These applications can be very useful for locating colleagues and assets in a busy office environment. The general consensus within the laboratory is that a larger deployment would increase the usage of such applications further.

The cost in walking to a neighbouring office to see who is there is not great enough to make it worth calling up a tracking application to check. A coverage zone extending over many floors means that the the cost of 'manually' checking whether a colleague is in their office two floors below seems too large in comparison to checking a location-aware map, and we would expect adoption of mapping applications to be much greater. It

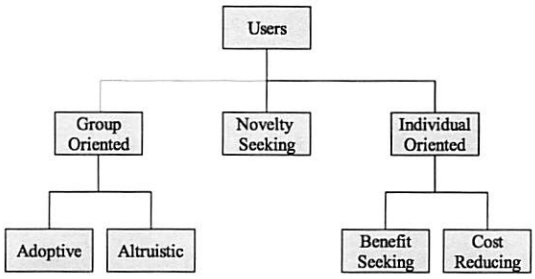


Figure 16: Classifying users of the Bat system

is true to say that in the three storey building that housed AT&T Research, Bat usage was far more common, perhaps attributable to the need to locate people quickly over a larger volume. Similarly, three responses to the survey indicated that a larger coverage area (or the coverage of physically separated areas) would encourage them to wear their Bat more often. The true worth of some location-aware applications will only become apparent with a very large deployment, beyond any implementation so far.

One popular application quoted in the survey was an alert service for when fresh coffee was available. This has no location aspect to it all — it uses the ability of the Bats to provide aural feedback to use them as wireless pagers. So it is interesting to find that the addition of more traditional applications of wireless technology has encouraged the adoption of Bats, simultaneously discouraging the pervasive application vicious cycle.

5.3 User classification

Mansley et al. grouped users of the system according to their common aims and tasks [9]. Over time it has become apparent that users of the Bat system at any instant can be be more generally classified according to how they both perceive and use the Bat system and its associated services. The classifications exist within a hierarchy (illustrated in Figure 16):

Novelty Seeking Users in this category wear their Bat and allow themselves to be tracked purely for novelty. Their main interest is in discovering the system capabilities and evaluating how it can enhance their lives, if at all.

Group oriented Users in this category are concerned with all other users in the system and can subdivided as follows:

Adoptive users These users adopt the system and use its applications on a regular basis. The majority of these are those researching location-aware computing.

Altruistic users These users derive little use from the Bat system, but are prepared to wear their Bats regularly to increase the system utility for other users. For example, they are willing to be tracked so others can locate them using Bat applications, but use more traditional methods when seeking those colleagues themselves.

Individual oriented These users view the system as it applies to them as individuals rather than group members. Within this category we find:

Benefit-seeking users These users wear a Bat if there is a personal benefit. For example, so a particular application they find useful will function.

Cost-reducing users The inverse of benefit-seeking users, these users will reject being tracked until a particular cost is removed, regardless of benefits. An example might be a user who will not be tracked because she feels she is being spied upon.

Figure 17 illustrates the difference in Bat usage for a benefit-seeking and an adoptive user. The data was collected across one month and clearly shows the benefit-seeker using the system sporadically, when it suits them. This contrasts with the adoptive user, who wears a Bat even when not using location-aware applications. We have very few examples of cost-reducing users within the laboratory, although a few have identified the cost of privacy to be a major issue. Indeed, in demonstrating the system to a wide variety of people the authors have become aware that the initial response of many is to fear a privacy invasion. The most common privacy fear quoted is that of the employer monitoring the employee. Interestingly, then, the survey found that privacy concerned laboratory users very little (Figure 18). This is perhaps attributable to two factors: firstly, the location data is not available outside the laboratory and so little information is available that could not be ascertained by walking through the laboratory; secondly, a Bat can be taken off at any time for privacy. It is also worth noting that, as a research establishment, working hours are not strict so there is little worry of them being policed. In a more commercial environment this may be more of a concern.

The survey of how often users wear Bats (Figure 8) is consistent with the empirical observation that most laboratory users are now either adoptive or benefit-seeking in nature. The peaks observable in Figure 10 occur at significant times. In particular, we see increased usage in late 2003, which we attribute to the roll-out of location-aware applications on the broadband phone network, and

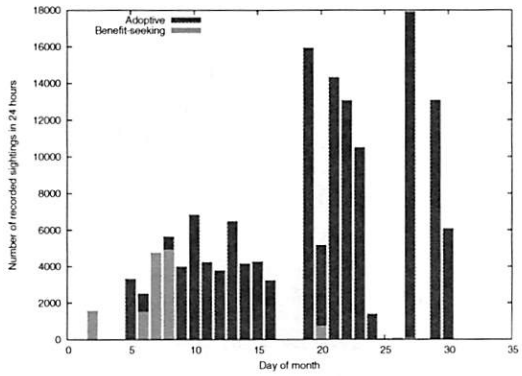


Figure 17: Daily sightings for two users over one month

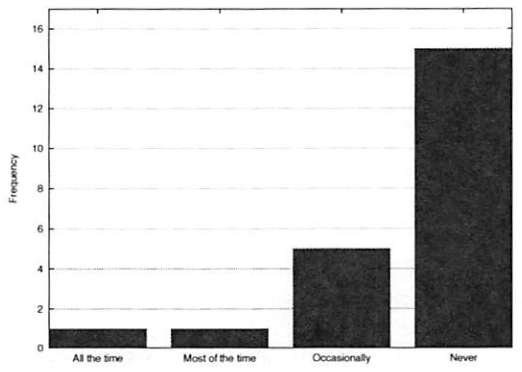


Figure 18: How much privacy concerns users

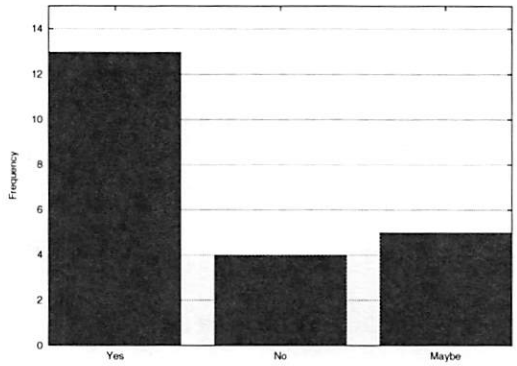


Figure 19: Ascertaining whether users would wear a Bat if it made the system more useful to others

some usage peaking around April 2004, when a further set of location-aware applications were made available.

When questioned about whether they would wear a Bat if it benefited their colleagues, 59% indicated that they would (Figure 19). Certainly, a higher proportion of users wear their Bats for limited periods when so requested for research or demonstration purposes. These facts indicate that many members can also be classified as altruistic in their Bat usage.

6 Conclusions

This paper has presented a unique portrayal of the deployment and usage of a location-aware system encompassing a medium-sized volume and regularly available to a wide range of technical people over an extended period.

Deployment experiences have produced a set of non-obvious guidelines to assist future large-scale deployments:

- Design for space usage rather than physical space.
- Beware the coincidence of physical and symbolic boundaries.
- Incorporate autonomous maintenance.
- Design for maximal signal penetration.
- Beware false positives and false negatives.
- Include feedback capabilities.
- Include power management.

This paper has also presented the results of surveying members of the laboratory that have access to Bats. The major highlights are:

- The majority (77.3%) feel comfortable wearing a 50g tag around their neck.
- Initial assignment of a location tag often results in a novelty period that typically lasts days.
- 42.1% now wear a Bat on a daily basis.
- The majority characterise their Bat as useful or fun.
- Applications are crucial to a good deployment
- A pervasive application vicious cycle is evident, where users reject some applications because others do not use them.
- False negatives in tracking complicate location-aware applications.
- Users differ in how they perceive and use the location-aware system, and distinct classifications are apparent.
- Location privacy is rarely a concern in the deployment environment.

The survey presented cannot be taken as definitive: it represents a small sample space and relates to a research (office) environment with a majority of technically competent users. However it is representative of the observed usage of the system since deployment. It would be interesting to compare these results for a series of deployments across areas with different typical usages, and also to investigate whether different sexes perceive and use the technology in different ways. The present deployment has too few female users to draw useful conclusions (note that this is due to a male dominance in the field rather than women opting out).

7 Future Work

The work has highlighted a need for a greater diversity of location-aware applications to allow users to determine what the technology can offer them. It has also underlined how important it is to test such technology on a larger scale than a single room: new challenges become apparent (maintaining the world model, aesthetics, etc.) and usage can be evaluated by regular users of the space to see what it actually contributes.

It is hoped that this paper will encourage large deployments of location technology in spaces outside research establishments, and lead to detailed human factors studies.

8 Acknowledgements

Many of the findings herein are the fruits of discussions with a variety of members of the Laboratory for Communication Engineering, who we thank for their contributions. We also wish to thank the MobiSys reviewers and our paper shepherd, Anthony LaMarca, for their useful comments.

References

[1] P. Bahl, V. N. Padmanabhan, and A. Balachandran. Enhancements to the RADAR user location and tracking system. Technical report, Microsoft Research, February 2000.

[2] A. Bystrom, T. Dahlroth, J. Eriksson, L. Fernandez, D. Holmgren, T. Johansson, P. Larsson, I. Styf, M. Stahl, N. Varillas, and M. Wu. Advanced Wavelan Positioning System. SMD116 Program Project, May 2001.

[3] R. K. Harle and A. Hopper. Building World Models By Ray-tracing Within Ceiling-Mounted Positioning Systems. In *Proceedings of UbiComp, Seattle, Washington, US*, October 2003.

[4] R. K. Harle and A. Hopper. Dynamic World Models from Ray-tracing. In *Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications, Orlando, Florida (PerCom 2004)*, March 2004.

[5] R.K. Harle and A. Hopper. Using Personnel Movements For Indoor Autonomous Environment Discovery. In *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications, Fort Worth, TX, US (PerCom 2003)*, pages 125–132, March 2003.

[6] A. Harter and A. Hopper. A distributed location system for the active office. *IEEE Network*, 8(1), 1994.

[7] M. Hazas and A. Ward. A Novel Broadband Ultrasonic Location System. In *Proceedings of UbiComp 2002: Fourth International Conference on Ubiquitous Computing*, 2002.

[8] J. Krumm, S. Harris, B. Meyers, B Brumitt, M. Hale, and S.. Shafer. Multi-Camera Multi-Person Tracking for EasyLiving. In *Proceedings of the Third International Workshop on Visual Surveillance*, July 2000.

[9] K. Mansley, A. Beresford, and D. Scott. The Carrot Approach: Encouraging use of location systems.

In *Proceedings of UbiComp*. Springer, September 2004.

[10] N. B. Priyantha, A. Chakraborty, and H. Balakrishnan. The Cricket Location-Support System. *Proceedings of the Sixth Annual ACM International Conference on Mobile Computing Networking*, August 2000.

[11] W. Rungsarityotin and T. Starner. Finding location using omnidirectional video on a wearable computing platform. In *ISWC*, pages 61–68, 2000.

[12] R. Want, A. Hopper, V. Falcao, and J. Gibbons. The Active Badge Location System. *ACM Transactions on Information Systems*, January 1992.

[13] R. Want, B. N. Schilit, N. I. Adams, R. Gold, K. Petersen, D. Goldberg, J. R. Ellis, and M. Weiser. An overview of the PARCTAB ubiquitous computing experiment. *IEEE Personal Communications*, 2(6):28–33, December 1995.

[14] A. M. R. Ward. *Sensor-driven Computing*. PhD thesis, Cambridge University, August 1998.

[15] M. A. Youssef, A. Agrawala, and A. U. Shankar. WLAN Location Determination via Clustering and Probability Distributions. In *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications, Fort Worth, TX, US (PerCom 2003)*, pages 143–152, mar 2003.

Accuracy Characterization for Metropolitan-scale Wi-Fi Localization

Yu-Chung Cheng[†‡] Yatin Chawathe[†] Anthony LaMarca[†] John Krumm[*]

[†]Intel Research, Seattle
{yatin.chawathe, anthony.lamarca}@intel.com

[‡]UC San Diego
ycheng@cs.ucsd.edu

[*]Microsoft Research
jckrumm@microsoft.com

Abstract

Location systems have long been identified as an important component of emerging mobile applications. Most research on location systems has focused on precise location in indoor environments. However, many location applications (for example, location-aware web search) become interesting only when the underlying location system is available ubiquitously and is not limited to a single office environment. Unfortunately, the installation and calibration overhead involved for most of the existing research systems is too prohibitive to imagine deploying them across, say, an entire city. In this work, we evaluate the feasibility of building a wide-area 802.11 Wi-Fi-based positioning system. We compare a suite of wireless-radio-based positioning algorithms to understand how they can be adapted for such ubiquitous deployment with minimal calibration. In particular, we study the impact of this limited calibration on the accuracy of the positioning algorithms. Our experiments show that we can estimate a user's position with a median positioning error of 13–40 meters (depending upon the characteristics of the environment). Although this accuracy is lower than existing positioning systems, it requires substantially lower calibration overhead and provides easy deployment and coverage across large metropolitan areas.

1 Introduction

A low-cost system for user devices to discover and communicate their position in the physical world has long been identified as a key primitive for emerging mobile applications. To this end, a number of research projects and commercial systems have explored mechanisms based on ultrasonic, infrared and radio transmissions [24, 29, 2, 5]. Despite these efforts, building and deploying location-aware applications that are usable by a wide variety of people in everyday situations is arguably no easier now than it was ten years ago.

Most current location systems do not work where people spend much of their time; coverage in these systems is either constrained to outdoor environments or limited to a particular building or campus with installed location infrastructure. For example, the most common location system, GPS (Global Positioning System) works worldwide, but it requires a clear view of its orbiting satellites. It does not work indoors and works poorly in many cities where the so called "urban canyons" formed by buildings prevent GPS units from seeing enough satellites to get a position lock. Ironically, that is exactly where many people spend the majority of their time.

Similarly, many of the research location systems such as RADAR [2], Cricket [24], and [10] only work in limited indoor environments and require considerable effort to deploy on a significantly larger scale. In indoor environments, these systems can provide accurate estimates of users' positions (within 2–4 meters). This accuracy comes at the cost of many hours of installation and/or calibration (e.g., over 10 hours for a 12,000 m^2 building [10]) and consequently has resulted in limited deployment. Arguably, for a large class of location-aware applications (for example, location-aware instant messaging or location-based search), ubiquitous availability of location information is crucial. The primary challenge in expanding the deployment of the above systems across, say, an entire city is the installation and calibration cost.

In this paper, we explore the question of how accurately a user's device can estimate its location using existing hardware and infrastructure and with minimal calibration overhead. This work is in the context of the Place Lab research project [18] where we propose a positioning infrastructure designed with two goals in mind: (1) maximizing coverage across entire metropolitan areas, and (2) providing a low barrier to entry by utilizing pre-deployed hardware. Unlike GPS, Place Lab works both indoors and outdoors. It relies on commodity hardware such as 802.11 access points and 802.11 radios built into users' devices to provide client-side positioning. Like some of the above systems, Place Lab works by having a client device listen for radio beacons in its environment and uses a pre-computed map of radio sources in the environment to localize itself.

An important tradeoff while deploying such a wide-area location system is the accuracy of the positioning infrastructure versus the calibration effort involved. Place Lab makes an explicit choice to minimize the

calibration overhead while maximizing coverage of the positioning system. We rely on user-contributed data collected by *war driving*, the process of using software [14, 20] on Wi-Fi and GPS equipped mobile computers and driving or walking through a neighborhood collecting traces of Wi-Fi beacons in order to map out the locations of Wi-Fi access points. A typical war drive around a single city neighborhood takes less than an hour. Contrast this with the typical calibration time for a single in-building positioning system that can require many hours of careful mapping. Moreover, war driving is already a well-established phenomenon with websites such as *wigle.net* gathering war drives of over 1.4 million access points across the entire United States.

Certainly, with limited calibration, Place Lab will estimate a user's location with lower accuracy. While this precludes using Place Lab with some applications, there is a large class of applications that can utilize high-coverage, coarse-grained location estimates. For example, resource finding applications (such as finding the nearest copy shop or Chinese restaurant) and social rendezvous applications have accuracy requirements that can be met by Place Lab even using limited calibration data.[1] Place Lab makes the tradeoff of providing localization on the scale of a city block (rather than a couple of meters), but manages to cover entire cities with significantly less effort than traditional indoor location systems.

Moving Wi-Fi location out of controlled indoor environments into this larger metropolitan-scale deployment is not as simple as just making the algorithms work outside and inside. The calibration differences demand a careful examination of the performance of positioning techniques in this new environment. In this paper, we evaluate the estimation accuracy of a number of different algorithms (many of which were originally proposed in the context of precise indoor location) [2, 15, 13] in this wider context with substantially less calibration data. Our contribution is two-fold: First, we demonstrate that it is indeed feasible to perform metropolitan-scale location with reasonable accuracy using 802.11-based positioning. Our experiments show that Place Lab can achieve accuracy in the range of 13–40 meters. Although this is nowhere near the accuracy of some indoor positioning systems, it is sufficient for many applications [3, 28, 7, 30]. Second, we compare a number of location algorithms and report on their performance in a variety of settings, for example, how the performance changes as the war-driving data ages, when the calibration data is noisy, or as the amount of calibration data is reduced.

The rest of the paper is organized as follows. In Section 2, we discuss relevant related work. Section 3 gives an overview of our research methodology. In Section 4, we present our experimental results. Finally, we discuss the implications of our results in Section 5 and conclude in Section 6.

2 Related Work

Location sensing for ubiquitous computing has been an active area of research since the PARCTAB [27] of 1993. Since then, many location technologies have been explored, most of them summarized in [11]. This paper is focused on using Wi-Fi as a location signal, an idea first published by Bahl and Padmanabhan in 2000 and called RADAR [2]. RADAR used Wi-Fi "fingerprints" previously collected at known locations inside a building to identify the location of a user's device down to a median error of 2.94 meters. Since then, there have been many other efforts aimed at using Wi-Fi for location. While nearly all Wi-Fi location work has been for inside venues, a few are intended to work outdoors as well. UCSD's Active Campus project [8] uses Wi-Fi to locate devices inside and outside based on a simplistic algorithm that relies on known positions of access points on a university campus. Recently, the Active Campus project has redesigned its system to use Place Lab instead of their original positioning system.

The main difference between Place Lab and previous Wi-Fi location work is that previous work has taken advantage of limited-extent venues where either the access point locations were known (e.g., ActiveCampus) or where extensive radio surveying was deemed practical (e.g., RADAR). Place Lab instead depends on war driving collected by a variety of users as they move naturally throughout a region. This means that the Wi-Fi radio surveys rarely contain enough data from any one location to compute meaningful statistics, thus eliminating the possibility of sophisticated probabilistic methods such as used in [10, 12, 16, 17].

As mentioned above, Place Lab is intended for metropolitan-scale deployment. Other large-scale location systems include satellite-based GPS and its variants, the Russian GLONASS and the upcoming European GALILEO systems. Place Lab differs from these in that it can work wherever Wi-Fi coverage is available, both indoors and outdoors, whereas satellite-based systems only work outdoors and even then only when they have a clear line of sight to the sky.

In the U.S., future requirements for cell phones demand that they be able to measure their own location to within 50–100 meters [4]. Other outdoor technologies include Rosum's TV-GPS [25], which is based on existing standards for digital TV synchronization signals and

[1] Dodgeball.com, for example, hosts a cellphone-based social meet-up application with thousands of daily users and relies on zipcodes to represent users' locations. It is well within Place Lab's ability to accurately estimate zip code.

gives mean positioning errors ranging from 3.2 to 23.3 meters in tests. The use of commercial FM radio signal strengths for location was explored in [15], which showed accuracies down to the suburb level.

As more research effort is devoted to Wi-Fi location, it becomes increasingly important to compare algorithms fairly on common data sets taken under known conditions. The only previous effort in this regard of which we are aware is [26], which compared three different Wi-Fi location algorithms in a single-floor office building. They compared a fingerprinting method similar to RADAR against two methods based on estimated signal strength probability distributions: histograms and Gaussian kernels. The histogram method performed slightly better than the other two. The paper made limited tests with varying the number of access points. Our paper undertakes more extensive testing using data from three different venues and tests against wide variations in density of known access points, density of the input mapping data sets, and noise in calibration data.

3 Methodology

All of our Wi-Fi-based positioning algorithms depend on an initial *training* phase. This involves war driving around a neighborhood with a Wi-Fi-enabled laptop and an attached GPS device. The Wi-Fi card periodically "scans" its environment to discover wireless networks while the GPS device records the latitude-longitude coordinates of the war driver when the scan was performed. We used active scanning where the laptop periodically broadcasts an 802.11 probe request frame and listens for probe responses from nearby access points.[2] Thus, a training data set is composed of a sequence of *measurements*: each measurement contains a GPS coordinate and a Wi-Fi scan composed of a sequence of *readings*, one per access point heard during the scan. Each reading records the MAC address of the access point and the signal strength with which it was heard.

Once the training phase is completed, the training data is processed to build a "radio map" of the neighborhood. The nature of this map depends on the positioning algorithm used. With a pre-computed radio map, a user's device can simply perform a Wi-Fi scan and position itself by comparing the set of access points heard against the radio map. We term this the *positioning* phase.

One question worth asking is "if GPS is insufficient as a positioning system in urban environments, how do we justify its use in constructing the training data sets." This is because, unlike GPS which requires line of sight to the sky, 802.11 radio beacons can be heard indoors as well as outdoors. Although our calibration data is collected using GPS entirely outdoors, we can still use it to position the user when he or she is indoors. Moreover, our use of a GPS device during the training phase does not imply that all users need to have a GPS device. For a given neighborhood, the training phase needs to be done only once by one user (until the AP deployment in that neighborhood changes). Once a neighborhood is mapped, all users can determine their position without needing any GPS device.

[2] Instead of active scanning, one could use passive scanning by listening on each Wi-Fi channel for beacon frames from the nearby access points.

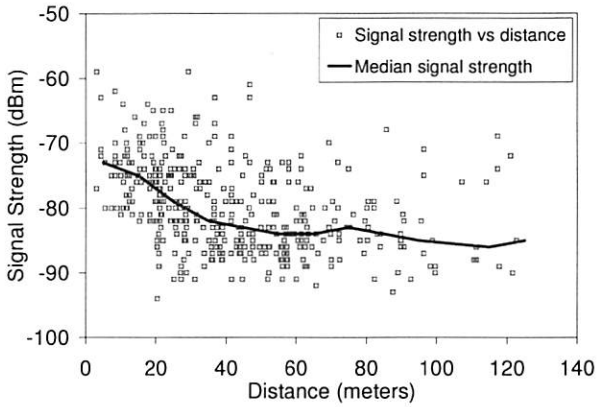


Figure 1: *(a) Measured signal strength (scatterplot and median values for 10 meter buckets) as a function of distance between the access point and a receiver. Signal strength with dBm values closer to zero means a stronger signal.*

3.1 Metrics for Positioning

Many previous radio-based positioning systems have used the observed signal strength as a indicator of distance from a radio source. In practice, this works only as well as the radio beacon's signal strength decays predictably with distance and is not overly-attenuated by factors such as the number of walls crossed, the composition of those walls, and multi-path effects. For instance, buildings with brick walls attenuate radio signals by a different amount than buildings made of wood or glass. In addition to fixed obstructions in the environment, people, vehicles and other moving objects can cause the attenuation to vary in a given place over time.

To characterize how observed 802.11 signal strengths varied with distance, we collected 500 readings at varying distances from a access points with well known locations in an urban area. Figure 1 illustrates the variation in signal strength for a single access point in a busy coffee shop[3] as a function of the distance between the AP and the receiver. This graph plots the individual readings as well as showing how the median signal strength changes with distance.

[3] We observed similar behavior from the other access points and we show one AP's data for the sake of clarity.

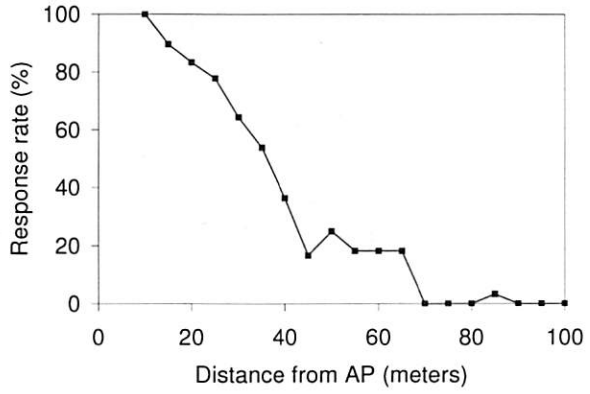


Figure 2: *Response rate for a single AP as a function of distance from that AP. Response rate is defined as follows: the percentage of times that a given AP was heard in all of the Wi-Fi scans at a specific distance from that AP. In the above graph we plot a histogram of response rate after grouping all distances into 5 meter buckets.*

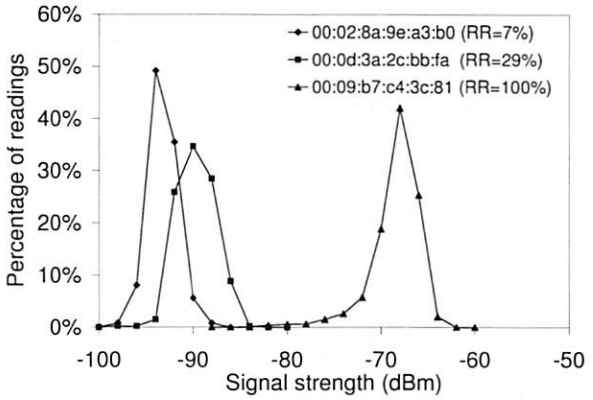


Figure 3: *Variation in signal strength over a one hour period for three distinct access points measured at a single location.*

The points for the individual readings show considerable spread for a given distance, and medium to weak readings (-70 to -90 dBm) occur at all distances. The curve showing the median signal strength does indicate however that there is a trend towards seeing weaker signals as distance from an access point grows. This indicates that while care needs to be taken, signal strength can be used as a weak indicator of distance and can thus be used to improve location estimation over simple observation.

In addition to signal strength, we explore a new metric for estimating a user's position. We define the *response rate* metric as follows: from the training data set, we collect all Wi-Fi scans that are at the same distance from an access point; we then compute the fraction of times that this AP is heard in that collection of Wi-Fi scans. For scans close to the AP, we expect the response rate to be high while for scans further away, with signal attenuation and interference, the AP is less likely to be heard and so the response rate will be low. We measured the response rate as a function of distance, and as can be seen in Figure 2, there is a strong correlation between the response rate and the distance from the AP. Our results may seem contradictory to the results from Roofnet [1], which shows packet loss rate has low correlation to distance. Roofnet measured the raw loss rate of 1500 byte broadcast packets across distant nodes (over 100 meters). On the other hand, we measure probe response rate of APs within 100 meters. The probe response packet is about 100 bytes and uses link-level unicast retransmissions. With shorter distance, smaller sized packets and link-level retransmissions, we noticed a stronger correlation between response rate and distance.

These observations tend to reduce our confidence in algorithms that depend on signal strength varying predictably as a function of distance to the access point. Response rate appears to vary much more predictably vs. distance. However, even though the effect of distance is largely unpredictable, Figure 3 indicates that for a given location, signal strengths are relatively stable. Thus different locations may still be reliably identified by their signal strength signature. We test these beliefs experimentally in the remainder of this paper.

3.2 Positioning Algorithms

Based on the above observations, we looked at three classes of positioning algorithms. In this section, we present an overview of these algorithms.

3.2.1 Centroid

This is the simplest positioning algorithm. During the training phase, we combine all of the readings for a single access point and estimate a geographic location for the access point by computing the arithmetic mean[4] of the positions reported in all of the readings. Thus, the radio map for this algorithm has one record per access point containing the estimated position of that AP.

Using this map, the centroid algorithm positions the user at the center of all of the APs heard during a scan by computing an average of the estimated positions of each of the heard APs. In addition to the simple arithmetic mean, we also experimented with a weighted version of this mechanism, where the position of each AP was weighted by the received signal strength during the scan.

[4]We do not in fact compute the centroid, but we still name this method as such instead of using the term "mean" so as to disambiguate the heuristic from other uses of arithmetic mean during the discussion of our experimental results.

3.2.2 Fingerprinting

This algorithm is based on an indoor positioning mechanism proposed in RADAR [2]. The hypothesis behind RADAR is as follows: at a given point, a user may hear different access points with certain signal strengths; this set of APs and their associated signal strengths represents a *fingerprint* that is unique to that position. As can be inferred from Figure 3, radio fingerprints are potentially a good indicator of a user's position. We used the same basic fingerprinting technique, but with the much coarser-grained war driving compared to the in-office dense dataset collected for RADAR. Thus, for fingerprinting, the radio map is the raw war-driving data itself, with each point in the map being a latitude-longitude coordinate and a fingerprint containing a scan of Wi-Fi access points and the signal strength with which they were heard.

In the positioning phase, a user's Wi-Fi device performs a scan of its environment. We compare this scan with all of the fingerprints in the radio map to find the fingerprint that is the closest match to the positioning scan in terms of APs seen and their corresponding signal strengths. The metric that we use for comparing various fingerprints is *k-nearest-neighbor(s) in signal space* as described in the original RADAR work. Suppose the positioning scan discovered three APs A, B, and C with signal strengths (SS_A, SS_B, SS_C). We determine the set of recorded fingerprints in the radio map that have the same set of APs and compute the distance in signal space between the observed signal strengths and the recorded ones in the fingerprints. Thus, for each matching fingerprint with signal strengths (SS'_A, SS'_B, SS'_C), we compute the Euclidean distance:

$$\sqrt{(SS_A - SS'_A)^2 + (SS_B - SS'_B)^2 + (SS_C - SS'_C)^2}$$

To determine the user's position, we pick the k fingerprints with the smallest distance to the observed scan and compute the average of the latitude-longitude coordinates associated with those k fingerprints. Based on preliminary experiments with varying values of k, we discovered that $k = 4$ provides good accuracy.

To account for APs that may have been deployed after the radio map was generated and for lost beacons from access points, we apply the following heuristics. First, during the positioning phase, if we discover any AP that never appears in the radio map, we ignore that AP. Second, when matching fingerprints to an observed scan, if we cannot find fingerprints with the same set of APs as heard in the scan, we expand the search to look for fingerprints that have supersets or subsets of the APs in the observed scan. We match fingerprints that have at most p different APs between the fingerprint in the radio map and the observed scan. These heuristics help improve the matching rate for fingerprints significantly from 70% to 99%. Across all of our data sets, we found that $p = 2$ provides the best matching rate without reducing overall accuracy.

Fingerprinting is based on the assumption that the Wi-Fi devices used for training and positioning measure signal strengths in the same way. If that is not the case (due to differences caused by manufacturing variations, antennas, orientation, etc.), one cannot directly compare the signal strengths. To account for this, we implemented a variation of fingerprinting called *ranking* inspired by an algorithm proposed for the RightSpot system [15]. Instead of comparing absolute signal strengths, this method compares lists of access points sorted by signal strength. For example, if the positioning scan discovered $(SS_A, SS_B, SS_C) = (-20, -90, -40)$, then we replace this set of signal strengths by their relative ranking, that is, $(R_A, R_B, R_C) = (1, 3, 2)$. Likewise, if $(SS'_A, SS'_B, SS'_C) = (-30, -15, -45)$, then $(R'_A, R'_B, R'_C) = (2, 1, 3)$. We compare the relative rankings using the Spearman rank-order correlation coefficient [23]:

$$r_S = \frac{\sum_i (R_i - \overline{R})(R'_i - \overline{R'})}{\sqrt{\sum_i (R_i - \overline{R})^2 \sum_i (R'_i - \overline{R'})^2}}$$

where $\overline{R}$ and $\overline{R'}$ are the means of the rank vectors. The Spearman coefficient ranges from $[-1, 1]$, with higher values indicating more similar rankings. Using the relative order of signal strengths in this way means that fingerprints will still match well in spite of differences in scale, offset, or any monotonically increasing function of signal strength separating the Wi-Fi devices. To use ranking, the value of r_S is substituted for the Euclidean distance in the fingerprint algorithm, and r_S is negated to make more similar ranks give a smaller number.

3.2.3 Particle Filters

Particle filters have been used in the past, primarily in robotics, to past fuse a stream of sensor data into location estimates [9, 21, 13]. A particle filter is a probabilistic approximation algorithm that implements a Bayes filter [6]. It represents the location estimate of a user at time t using a collection of weighted *particles* $p_t^i, w_t^i, (i = 1...n)$. Each p_t^i is a distinct hypothesis about the user's current position. Each particle has a weight w_t^i that represents the likelihood that this hypothesis is true, that is, the probability that the user's device would hear the observed scan if it were indeed at the position of the particle. A detailed description of the particle filter algorithm can be found in [13].

Particle-filter based location techniques require two input models: a *sensor model* and a *motion model*. The

Neighborhood	AP density (APs/km^2)	# APs/ scan
Downtown Seattle	1030	2.66
Ravenna	1000	2.56
Kirkland	130	1.41

Table 1: *AP density in the three areas measured per square kilometer and per Wi-Fi scan. The # APs/scan includes only those Wi-Fi scans that detected at least one access point.*

sensor model is responsible for computing how likely an individual particle's position is, given the observed sensor data. For Place Lab, the sensor model estimates how likely it is that a given set of APs would be observed at a given location. The motion model's job is to move the particles' locations in a manner that approximates the motion of the user.

For our experiments, we built two sensor models: one based on signal strength, while the other based on the response rate metric defined earlier. During the training phase, for each access point, we build an empirical model of how the signal strength and response rate vary by distance. Rather than fit the mapping data to a parametric function, Place Lab maintains a small table with an entry for each 5 meter increment in distance from the estimated AP location. Response rates are recorded as a percentage, while the signal strength distribution is recorded as a fraction of observations with low, medium and high strength for each distance bucket. The low/medium/high cut-offs are determined empirically to split the training data for each AP uniformly into the three categories. As an example, Place Lab will compute and record that for an AP x, its response rate at 60 meters is (say) 55%. We also record that at 60 meters, that AP will be seen with medium signal strength much more often than low or high. Given a new Wi-Fi scan, the sensor model determines a particle's likelihood as follows: for each AP in the scan, we look up the response rate or the probability of seeing the measured signal strength based on the distance between the particle and the estimated AP location in the radio map.

Place Lab uses a simple motion model that moves particles random distances in random directions. Our future work includes incorporation of more sophisticated motion models (such as those by Patterson et al. [22]) that model direction, velocity and even mode of transportation.

3.3 Data Collection

We collected traces of war-driving data in three neighborhoods in the Seattle metropolitan area[5]:

- Downtown Seattle: a mix of commercial and residential urban high-rises
- Seattle's Ravenna neighborhood: a medium-density residential neighborhood
- Kirkland, Washington: a sparse suburb of single-family homes

For each neighborhood, we collected traces in two phases. First we constructed training data sets by driving around each neighborhood for thirty minutes with a Wi-Fi laptop and a GPS unit. Our data collection software performed Wi-Fi scans four times per second using an Orinoco-based 802.11b Wi-Fi card. GPS readings were logged approximately once per second. To assign latitude-longitude coordinates to each Wi-Fi scan, we used linear interpolation between consecutive GPS readings based on the timestamps associated with the Wi-Fi scans and the two GPS readings.

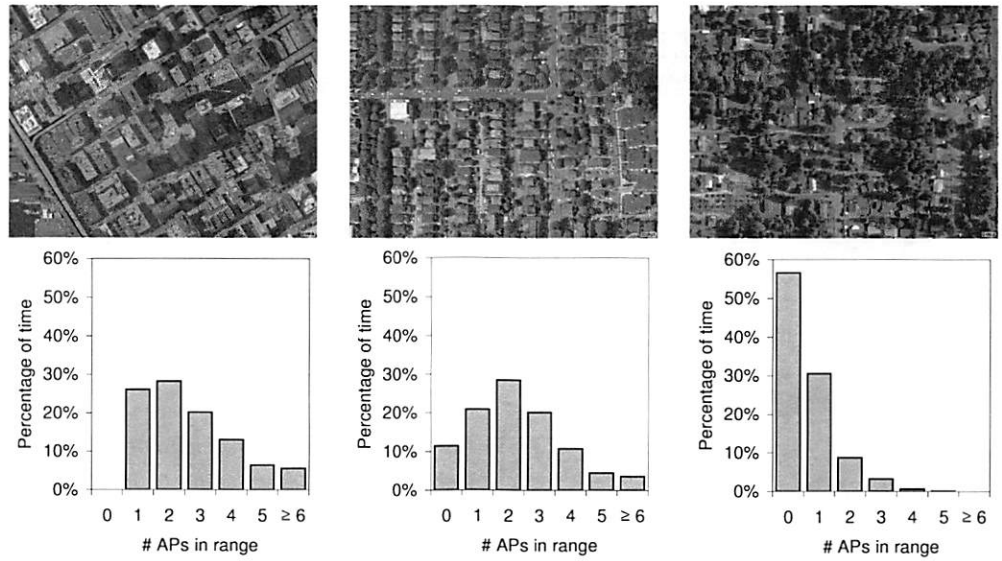
In the second phase, we collected another set of traces for each neighborhood. These traces are used as test data to estimate the positioning accuracy of the various Place Lab algorithms. In this phase, Place Lab used only the Wi-Fi scans collected in the trace, while the GPS readings were used as "ground truth" to compute the accuracy of the user's estimated position. To ensure that we gathered clean ground truth data, we tried to navigate within areas where the GPS device continuously reported a GPS lock and eliminated (and re-gathered) traces where the GPS data was obviously erroneous. Note that GPS has an accuracy of 5–7 meters, bounding the accuracy of our measurements to this level of granularity.

Even though Place Lab can be used both outdoors and indoors, most of our evaluation below is based on traces that were collected entirely outdoors. This limitation is due to the fact that we use GPS as ground truth and hence can evaluate Place Lab's performance only when GPS is available. In Section 4.6, we will present results from a simple experiment where we used Place Lab indoors.

4 Evaluation

In this section, we present our results from a suite of experiments conducted using the above data sets. Our results demonstrate the effect of varying a number of parameters on the accuracy of positioning using Place Lab.

[5] Although the Seattle metropolitan area is a tech-friendly environment and consequently has a higher proliferation of Wi-Fi access points than many other parts of the country (or the world), we believe that it represents the leading edge of an upward growth trend. So although results from these areas may not necessarily apply today to other cities with lower Wi-Fi coverage, we expect other metro areas will eventually match Seattle's current coverage density.



(a) Downtown (b) Ravenna (c) Kirkland

Figure 4: *Density of access points (and satellite photos provided through Microsoft's Terraserver) for the neighborhoods in which we ran our experiments.*

4.1 Analysis of trace data

Figure 4 shows the distribution of the number of access points in range per scan for each of the three areas we measured. Table 1 shows the density of APs in the three areas. As expected, we noticed the highest density of APs in the downtown urban setting with an average of 2.66 APs per scan and no scans without APs. Also not surprisingly, the suburban traces saw zero APs (that is, no coverage) more than half the time and rarely saw more than one. Interestingly, the residential Ravenna data had almost the same number of APs per km^2 as downtown. With the exception of the approximately 10% of scans with no coverage, the AP density distribution for Ravenna fairly closely matched the downtown distribution.

We also plotted the median and maximum range of each AP based on estimated positions of the AP from the radio map. Figure 5 shows a cumulative distribution function (CDF) of the median and maximum ranges for each of the three areas. In the relatively sparse Kirkland area, we notice that APs can be heard at a longer range than in Ravenna. We believe that this is due to the fact that Ravenna has a much denser distribution of access points and thus experiences higher interference (and consequently shorter range) as the data-gathering station gets further away from the AP. On the other hand, in downtown, which has about the same AP density as Ravenna, the maximum AP range is higher. This is due to the large number of tall buildings in downtown; APs that are located on higher floors often have a much longer range.

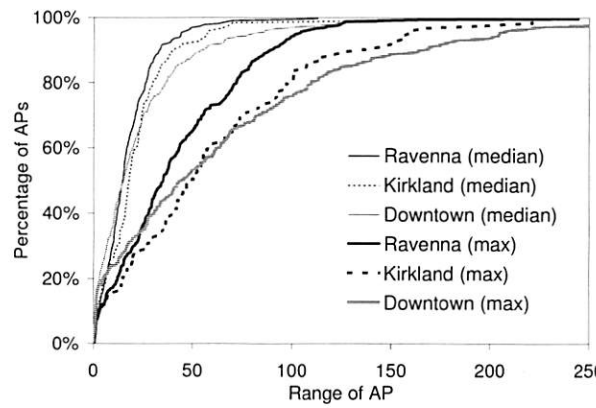


Figure 5: *CDF showing the median and maximum range of APs in meters.*

4.2 Relative performance

We now compare the relative performance of our positioning algorithms across the three areas. To evaluate the accuracy of Place Lab's positioning, we compare the position reported by Place Lab to that reported by the GPS device during the collection of the positioning trace. Table 2 summarizes the results. Ravenna, with its high density of short-range access points, performs the best and can estimate the user's position with a median error of 13–17 meters. Surprisingly, for Ravenna, there is little variation across the different algorithms. Even the simple centroid algorithms perform relatively well.

On the other hand, in Kirkland, with much lower AP

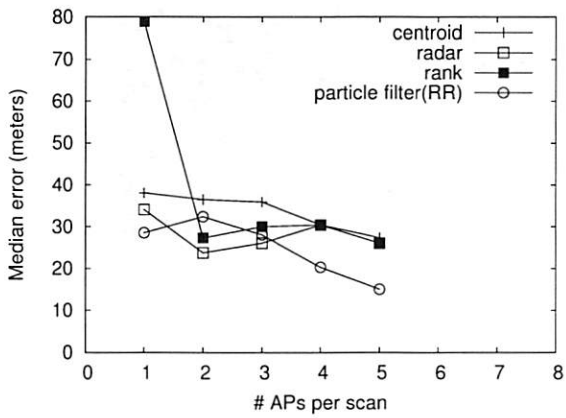


Figure 6: *Median error as a function of number of APs heard in a scan (Kirkland).*

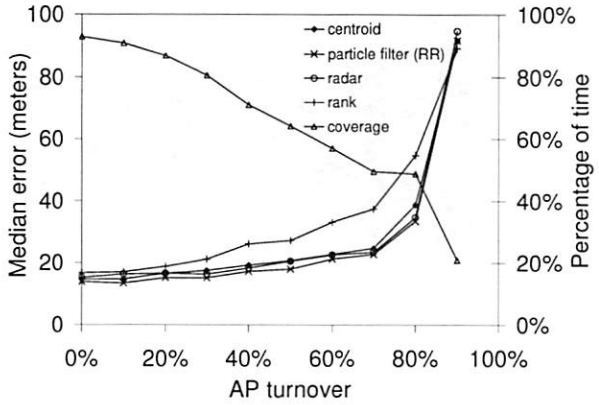


Figure 7: *Median error as a function of AP turnover (the percentage of APs that are deployed but are not part of the training data) for Ravenna. The coverage plot represents the variation in the percentage of time that at least one known AP was within range.*

density, we notice that the median positioning error is 2–3 times worse. But, as we move from the centroid algorithm to the particle-filter-based techniques, we notice a substantial improvement in accuracy (25% decrease in median error). In this case, the smarter algorithms were better able to filter through the sparse data and estimate the user's position. However, one algorithm that performed quite poorly in Kirkland is ranking. This is because of the significant number of times that a Wi-Fi scan produced a single AP. With just one AP, there can be no relative ranking, and hence the algorithm picks a random set of k fingerprints that contain the AP and attempts to position the user at the average position of those randomly chosen fingerprints.

In the downtown area, even though the AP density is the same as Ravenna, median error is higher by 5–10 meters. This is due to the fact that APs in the tall buildings of downtown typically have a larger range and thus can be heard much further away than APs in Ravenna.

Finally, to understand the effect of the numbers of APs per scan on the positioning accuracy, we separated the output of the positioning traces based on the number of APs that were heard in each scan. For each group, we computed the median error. Figure 6 shows the median error as a function of the number of APs that were seen in a scan for one of the areas. Variants of algorithms that performed similar to their counterparts are left out for clarity. As we can see, the higher the number of APs heard during a scan, the lower the median positioning error. This graph also shows quite starkly the poor performance of ranking in the presence of a single known access point.

4.3 Effect of AP Turnover

When building a metropolitan-scale positioning system, an important question to ask is how fresh the training data needs to be in order to produce reasonable positioning estimates. In particular, new access points may be deployed while existing APs are decommissioned. We can express AP turnover as the percentage of currently deployed access points that are not present in the training data set. To measure the effect of such obsolete training data, we generated variations of the three training sets (one for each area) by randomly dropping access points from the original data. This simulates the effect of having performed the training war drive before these APs were deployed. Note that decommissioned APs are, for the most part, less of a problem. They result in dead state in the training data and, except for fingerprinting, their presence in the training data does not affect the positioning algorithm.

Figure 7 shows how AP turnover affects the median positioning error for the Ravenna neighborhood.[6] From the figure, we can see that as the AP turnover rate increases, the coverage (that is, the percentage of time that at least one known AP is within range of the user) decreases. What is important to note though is that for most algorithms, an AP turnover of even 30% produces minimal effect on the positioning accuracy. The positioning error starts to increase noticeably only after at least half of the access points in the area have turned over. The exception to this is the ranking algorithm. Since it relies on a fairly coarse metric for positioning (relative rank order of AP signal strengths), the fewer APs available for this relative ordering, the worse its performance.

This data suggests that training data for an area does not need to be refreshed too often. Of course, the refresh rate (and exactly when to refresh) would depend

[6]Other neighborhoods showed similar behavior and their data is left out for clarity

Algorithm		Downtown (meters)	Ravenna (meters)	Kirkland (meters)
centroid	basic	24.4	14.8	37.0
	weighted	23.4	14.5	37.0
fingerprint (k=4)	radar	18.5	15.3	30.0
	rank	20.3	16.7	59.5
particle filter	signal strength	18.0	14.4	29.7
	response rate	21.3	12.9	28.6

Table 2: Median error in meters for all of our algorithms across the three areas.

on the turnover rate of APs in that area. Our experiments used variations that randomly dropped access points across the original training data. This is reasonable in dense urban areas as well as in residential neighborhoods where there are many uncorrelated deployments of access points. This is less true of large-scale deployments, say across a suburban office complex or a university campus, that span an entire neighborhood and are typically upgraded in lock step.

Correlated turnover can be an issue even in seemingly uncorrelated neighborhoods. As an anecdotal example, we looked at the AP turnover in the Ravenna neighborhood, which happens to be located near the University of Washington. Some of the blocks that we war drove are home to the university fraternity houses. We compared our Ravenna data set (which was gathered after the start of the school year) to another data set of the same neighborhood that we gathered earlier during the middle of the summer. At least 50% of the APs in the later set of traces did not appear in our earlier traces. This was due to the fact that between our two experiments, new students had arrived for the fall quarter while summer students left en masse. Hence, when determining the schedule and frequency of refresh for the training data, it is important to take into account such social factors that can have a significant effect on the deployment of access points.

4.4 Effect of noisy GPS data

Some geographic regions will have higher GPS errors than others, due to urban canyons or foliage. This next experiment was designed to measure how robust the algorithms are to errors in the measured positions of the training data, which are normally measured via GPS. One way to assess the effect of poor GPS data would be to single out those areas where the GPS data is known to be poor. However, such locations may well exhibit Wi-Fi anomalies like multipath in urban canyons or RF blockage in areas of thick foliage [19]. In order to assess the overall effect of inaccurate GPS data on all our test data, we added zero mean Gaussian noise to the originally measured latitude-longitude readings in the training data. To make the results easier to understand, we

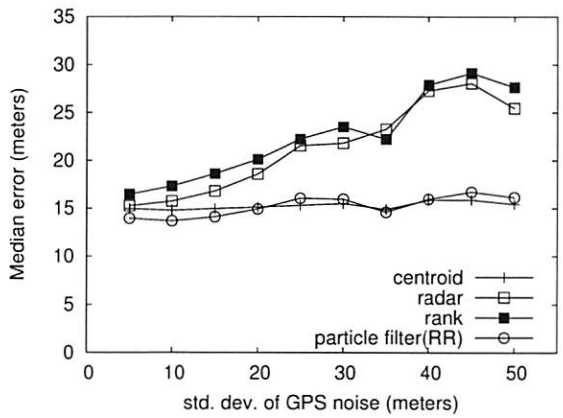


Figure 8: *Median positioning error as a function of the standard deviation of GPS noise (Ravenna).*

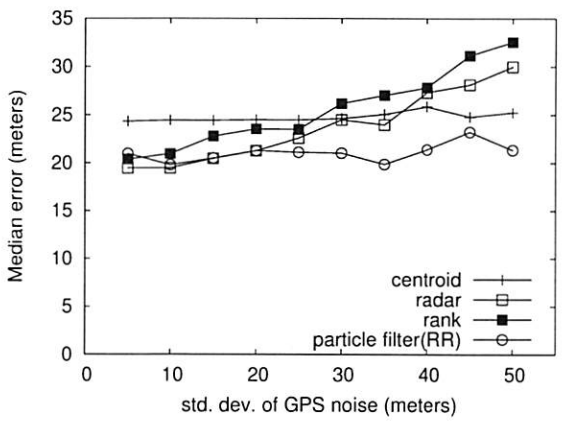


Figure 9: *Median positioning error as a function of the standard deviation of GPS noise (Downtown).*

specified the standard deviation of the noise in meters and converted to degrees latitude or longitude with the following:

$$\sigma_m = stddev\ in\ meters$$

$$\sigma_{lat} = \frac{180\sigma_m}{\pi r} = latitude\ stddev$$

$$\sigma_{lon} = \frac{180\sigma_m}{\pi r \cos(latitude)} = longitude\ stddev$$

$$r = 6371 \times 10^3 = mean\ earth\ radius\ (meters)$$

We used these modified training data sets to generate new radio maps and then computed the positioning error by running the unmodified test traces through these maps. Figure 8 shows the effect that GPS noise has on positioning accuracy. For the centroid algorithms, with sufficient number of observations, the GPS noise cancels out while creating the radio map. Hence we see no discernible effect on the performance of the centroid algorithm. Similarly, the particle filter techniques rely on empirical models built using the same radio map. Some error is introduced into these models because each of the readings that contribute to the histogram for the model has noisy positioning data, and thus can get misclassified. However the effect of this is not substantial as seen in Figure 8. On the other hand, for the fingerprinting techniques, the raw (and noisy) GPS positions are used directly in the radio map. Hence these techniques suffer the most in the presence of GPS noise. Figure 9 shows the same experiment for the Downtown data set. Similar trends are visible in this graph, as well as for the Kirkland data (not shown).

4.5 Density of mapping data

In the next experiment, we vary the geometric density of mapping points in the training data set. We expect the accuracy will go down with decreasing density of training data points. This variation is intended to find the effect of reduced density in the training set (which in turn means less calibration). We measured training density as the mean distance from each point (latitude-longitude coordinate) in the training data set to its geometrically nearest neighbor. In order to generate this variation, we first split the training file into a grid of 10m × 10m cells. We then eliminated Wi-Fi scans from the training data set one by one, creating a new mapping file after each elimination. To pick the next point to eliminate, we randomly picked one point in the cell with the highest population of points. If two or more cells tied for the maximum number of points, we picked between those tied cells at random. By eliminating points from the cells with the highest population, this algorithm tended to eliminate points around higher densities, driving the training files toward a more uniform density of data points for testing.

Figure 10 shows the median positioning error as a function of the average distance to the nearest neighbor in the training data set. The higher the average distance, the sparser the data set.[7] From the figure, we can see that until the mapping density drops below 10 meter average

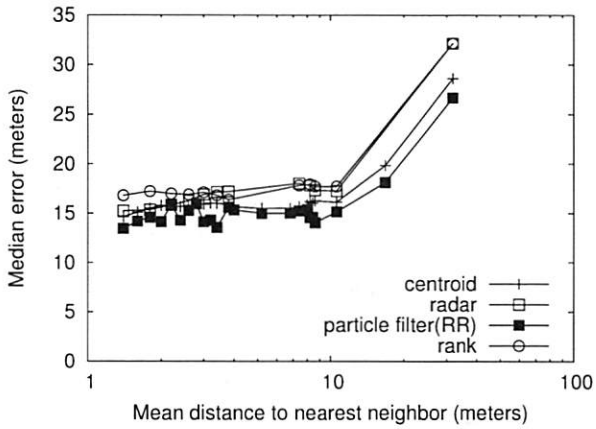


Figure 10: *Positioning error as a function of the average distance between points in the training data set (Ravenna). Note that the x-axis is a logarithmic scale.*

distance between points, there is no appreciable effect on median error. Beyond that point, however, the positioning error increases sharply. There is not much distinction in this behavior across the various algorithms. This suggests that a training map generated at a scanning rate of one Wi-Fi scan per second and a driving speed of 20–25 miles/hour[8] (or faster speeds with repeated drives through the same neighborhood) is sufficient to provide reasonable accuracy.

4.6 An Indoor Usage Scenario

All of the above experiments used training and positioning data sets that were collected entirely outdoors. Since the training data requires GPS, it must necessarily be collected outdoors. In the positioning phase, Place Lab can be used both outdoors and indoors. However, it is difficult to quantify the accuracy while indoors since we cannot collect any ground truth GPS data. To demonstrate the usability of Place Lab indoors, we ran a simple experiment where we collected two-minute-long positioning traces in nine different indoor locations, along with longer outdoor training traces around those locations. For each location, we computed the average position estimated by Place Lab. In addition, we determined average latitude-longitude positions for all nine locations by plotting their addresses into a mapping tool (MapPoint). Table 3 summarizes the error between Place Lab's average estimate and the latitude-longitude position from MapPoint.

The average error from this simple experiment ranges from 9 to 98 meters. Although at the high end the average error is significantly higher than in our previous experiments, it comes partially from inaccurate ground

[7] Note that some of the highest density data points in the graph arise as a result of the fact that we sometimes war drove the same streets more than once thus generating a denser training data set.

[8] 10 meters between scans is equivalent to a driving speed of 10 × 3600 meters/hour, that is, 22.5 miles/hour.

Location	Avg. error (meters)
Home 1	9.0
CS department	9.1
Downtown mall	9.5
Office	34.2
Bakery	38.9
Home 2	84.3
Doctor's office	85.2
Café	92.4
Home 3	98.7

Table 3: *Average error with Place Lab when used in indoor settings.*

truth data: we used a single latitude-longitude point to represent each location when in fact each building may be several tens of meters across. We stress that this experiment is not an attempt to draw general conclusions about the accuracy of Place Lab when used indoors with training data that was collected outdoors. The experiment only serves to show that unlike GPS, Place Lab works indoors (and when there is no line of sight to the sky). Of course, one should remember that the limited calibration associated with Place Lab inherently means that it cannot be used for precise indoor location applications.

4.7 Summary of Results

To summarize the results from the above experiments, we noted a few dominant effects that play a role in positioning accuracy using the various location techniques.

- The density of access points as well as the average range of APs in a region affect the positioning accuracy. In our experiments, we discovered that the Ravenna neighborhood with a dense collection of APs in low-rise buildings provided the best accuracy while suburban neighborhoods with sparse coverage showed the least accuracy.
- In dense urban environments, even a simple centroid-based positioning algorithm provides the same accuracy as more complex techniques. The complex techniques are more valuable in sparser environments with limited calibration data.
- Even a radio map that is old enough to have only 50% of the deployed APs is sufficient without degrading positioning accuracy by more than a few meters. Our mapping density experiments show that training data collected at the rate of one scan per second with a driving speed of 20–25 miles/hour is enough to build accurate radio maps.
- Noise in GPS data (which can result from either poor GPS units or due to urban canyons) affects fingerprint-based techniques much more than the other techniques. Thus, in environments where it is hard to collect accurate GPS data for training, we expect the particle-filter-based algorithms to perform better.
- The rank fingerprint algorithm was usually among the worst performing algorithms. Its poor performance was largely due to the fact that it throws away absolute signal strengths. However, in return, it also sheds its sensitivity to potential systematic differences in the way different Wi-Fi devices measure signal strength.

5 Discussion

We now discuss some of the practical issues that must be addressed when deploying such a system in the real world. Our experimental setup used *active* scanning to probe for nearby access points. However, APs can sometimes be configured to not respond to broadcast probe packets. Moreover, they may never send out broadcast beacon packets announcing their presence either. In such scenarios, passive scanning where the Wi-Fi card does not send out probe requests, and instead simply sniffs traffic on each of the Wi-Fi channels, may be used. Passive scanning relies on network traffic to discover APs and hence can detect even cloaked APs that do not necessarily advertise their network IDs.

All of our experiments were performed in outdoor environments since we needed access to GPS data even for our positioning traces to allow us to compare Place Lab's estimated positions to some known ground truth. However, Place Lab can work indoors as well. Even though we were unable to perform extensive experiments to measure Place Lab's accuracy when indoors, our regular use of Place Lab in a variety of indoor situations has shown that it can routinely position the user within less than one city block of their true position. As long as the user's device can hear access points that have been mapped out, Place Lab can estimate its position. This accuracy is not sufficient for indoor location applications that require room-level (or greater) accuracy, but is more than enough for other coarser-grained applications. For such applications, Place Lab can function as a GPS replacement that works both indoors and outdoors.

We used Wi-Fi cards based on the Orinoco chipset for all of our experiments. As pointed out in [10], different chipsets can report different signal strength values for the same AP at the same location. This is because each chipset interprets the raw signal strength value differently. However, there is a linear correlation for measured signal strengths across chipsets. This was first shown by Haeberlen et. al. [10]. We validated this claim for the following chipsets: Orinoco, Prism 2, Aironet, and Atheros. If we record the chipset information when

collecting data sets, then even if the positioning is done using a chipset that is different from the one used for collecting the training data, a simple linear transformation will be sufficient to map the signal strength values across them. Of course, this is important only for those algorithms that actually rely on signal strength for positioning.

As the first systematic study of metropolitan-scale Wi-Fi localization, our accuracy comparisons suggest what to pursue in terms of new positioning algorithms. We found that different algorithms work best with different densities and ranges of access points. Since both of these quantities can be measured beforehand, the best algorithm could be automatically switched in depending on the current situation. Further, the size of the radio map can be a substantial issue for small mobile devices. The centroid and particle filter radio maps are relatively compact whereas fingerprinting algorithms require the entire training data set as their radio map. If (say for privacy concerns), the user wishes to store the radio map for their region on their local device, this may be a factor in determining the appropriate algorithm to use.

Our studies also compare the effects of density of calibration data, noise in calibration, and age of data on the centroid, particle filter and fingerprinting algorithms. An algorithm combining these techniques is feasible, and it may prove to be both robust and accurate. Moreover, one could incorporate additional environmental data such as (for example) constrained GIS maps of city streets and highways for navigation applications to improve the positioning accuracy. While we leave these ideas for future work, our study gives a concrete basis for choosing what to do next.

6 Conclusion

Place Lab is an attempt at providing ubiquitous Wi-Fi-based positioning in metropolitan areas. In this paper, we compared a number of Wi-Fi positioning algorithms in a variety of scenarios. Our results show that in dense urban areas Place Lab's positioning accuracy is between 13–20 meters. In more suburban neighborhoods, accuracy can drop down to 40 meters. Moreover, with dense Wi-Fi coverage, the specific algorithm used for positioning is not as important as other factors including composition of the neighborhood (lots of tall buildings versus low-rises), age of training data, density of training data sets, and noise in the training data. In sparser neighborhoods, sophisticated algorithms that can model the environment more richly win out. All of this positioning accuracy (although lower than that provided by precise indoor positioning systems) can be achieved with substantially less calibration effort: half an hour to map out an entire city neighborhood as compared to over 10 hours for a single office building [10].

Interested readers may download the data sets used for these experiments and the Place Lab source code from http://www.placelab.org/.

References

[1] D. Aguayo, J. Bicket, S. Biswas, G. Judd, and R. Morris. Link-level measurements from an 802.11b mesh network. In *Proceedings of SIGCOMM '04*, Aug. 2004.

[2] P. Bahl and V. N. Padmanabhan. RADAR: An In-Building RF-Based User Location and Tracking System. In *Proceedings of IEEE Infocom 00*, April 2000.

[3] Dodgeball.com. Mobile social software. http://www.dodgeball.com/.

[4] E911 and E112 Resources. http://www.globallocate.com/.

[5] P. Enge and P. Misra. The Global Positioning System. *Proceedings of the IEEE (Special Issue on GPS)*, pages 3–172, January 1999.

[6] D. Fox, J. Hightower, L. Liao, D. Schulz, and G. Borriello. Bayesian filtering for location estimation. *IEEE Pervasive Computing*, 2(3):24–33, July-September 2003.

[7] Google.com. Google local: Find local businesses and services on the web. http://local.google.com/.

[8] B. G. Griswold et al. Using mobile technology to create opportunistic interaction on a university campus. In *Proceedings of UBICOMP Workshop on Supporting Spontaneous Interaction in Ubiquitous Computing Settings*, September 2002.

[9] F. Gustafsson, F. Gunnarsson, N. Bergman, U. Forssell, J. Jansson, R. Karlsson, and P.-J. Nordlund. Particle Filters for Positioning, Navigation and Tracking. *IEEE Transactions on Signal Processing*, 50:425–435, February 2002.

[10] A. Haeberlen, E. Flannery, A. M. Ladd, A. Rudys, D. S. Wallach, and L. E. Kavraki. Practical robust localization over large-scale 802.11 wireless networks. In *Proceedings of ACM MobiCom*, Philadelphia, PA, Sept. 2004.

[11] J. Hightower and G. Borriello. Location systems for ubiquitous computing. *IEEE Computer*, 34(8):57–66, August 2001.

[12] J. Hightower and G. Borriello. Accurate, Flexible, and Practical Location Estimation for Ubiquitous Computing. In *Proceedings of International Conference on Ubiquitous Computing (UBICOMP)*, 2004.

[13] J. Hightower and G. Borriello. Particle filters for location estimation in ubiquitous computing: A case study. In *Proceedings of International Conference on Ubiquitous Computing (UBICOMP)*, 2004.

[14] Kismet. http://www.kismetwireless.net/.

[15] J. Krumm, G. Cermak, and E. Horvitz. RightSPOT: A Novel Sense of Location for a Smart Personal Object. In *Proceedings of International Conference on Ubiquitous Computing (UBICOMP)*, October 2003.

[16] J. Krumm and E. Horvitz. Locadio: Inferring motion and location from wi-fi signal strengths. In *Proceedings of International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous'04)*, 2004.

[17] A. M. Ladd, K. E. Bekris, A. Rudys, G. Marceau, and L. E. Kavraki. Robotics-Based Location Sensing Using Wireless Ethernet. In *Proceedings of ACM MobiCom*, September 2002.

[18] A. LaMarca et al. Place lab: Device positioning using radio beacons in the wild. In *Proceedings of International Conference on Pervasive Computing (Pervasive)*, June 2005.

[19] C. Ma, G.-I. Jee, G. MacGougan, G. Lachapelle, S. Bloebaum, G. Cox, L. Garin, and J. Shewfelt. Gps signal degradation modeling. In *Proceedings of International Technical Meeting of the Satellite Division of the Institute of Navigation*, Sept. 2001.

[20] NetStumbler. http://www.netstumbler.com.

[21] P.-J. Norlund, F. Gunnarsson, and F. Gustafsson. Particle Filters for Positioning in Wireless Networks. In *Proceedings of EUSIPCO*, September 2002.

[22] D. J. Patterson, L. Liao, D. Fox, and H. Kautz. Inferring High-Level Behavior from Low-Level Sensors. In *Proceedings of International Conference on Ubiquitous Computing (UBICOMP)*, 2003.

[23] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C*. Cambridge University Press, 1992.

[24] N. B. Priyantha, A. Chakraborty, and H. Balakrishnan. The Cricket Location-Support System. In *Proceedings of ACM MobiCom'00*, July 2000.

[25] M. Rabinowitz and J. Spilker. A new positioning system using television synchronization signals. http://www.rosum.com/.

[26] T. Roos, P. Myllymaki, H. Tirri, P. Misikangas, and J. Sievanan. A probabilistic approach to wlan user location estimation. *International Journal of Wireless Information Networks*, 9(3), July 2002.

[27] B. N. Schilit, N. Adams, R. Gold, M. Tso, and R. Want. The PARCTAB Mobile Computing System. In *Proceedings of Workshop on Workstation Operating Systems*, pages 34–39, Oct. 1993.

[28] I. Smith et al. Social disclosure of place: From location technology to communication practices. In *Proceedings of International Conference on Pervasive Computing (Pervasive)*, May 2005.

[29] R. Want, A. Hopper, V. Falcao, and J. Gibbons. The Active Badge Location System. *ACM Transactions on Information Systems*, 1992.

[30] Yahoo! Inc. Yahoo local: Find businesses and services near you. http://local.yahoo.com/.

Energy Efficiency of Handheld Computer Interfaces: Limits, Characterization and Practice *

Lin Zhong and Niraj K. Jha
Department of Electrical Engineering
Princeton University
Princeton, NJ 08544
{lzhong, jha}@princeton.edu

Abstract

Energy efficiency has become a critical issue for battery-driven computers. Significant work has been devoted to improving it through better software and hardware. However, the human factors and user interfaces have often been ignored. Realizing their extreme importance, we devote this work to a comprehensive treatment of their role in determining and improving energy efficiency. We analyze the minimal energy requirements and overheads imposed by known human sensory/speed limits. We then characterize energy efficiency for state-of-the-art interfaces available on two commercial handheld computers. Based on the characterization, we offer a comparative study for them.

Even with a perfect user interface, computers will still spend most of their time and energy waiting for user responses due to an increasingly large speed gap between users and computers in their interactions. Such a speed gap leads to a bottleneck in system energy efficiency. We propose a low-power low-cost cache device, to which the host computer can outsource simple tasks, as an interface solution to overcome the bottleneck. We discuss the design and prototype implementation of a low-power wireless wrist-watch for use as a cache device for interfacing.

With this work, we wish to engender more interest in the mobile system design community in investigating the impact of user interfaces on system energy efficiency and to harvest the opportunities thus exposed.

I. Introduction

Energy consumption is a critical concern for battery-driven mobile devices, such as handhelds, laptops, and cell-phones. Most handheld computers serve their users directly through human-computer interaction, and most tasks are interactive. From the user's perspective, the concern is not really the power consumption itself but what the user can do, given the battery lifetime. Energy efficiency is, therefore, better evaluated in terms of energy consumption per user task. At a higher level, one needs to evaluate:

$$\frac{\text{User productivity}}{\text{Average power consumption}}$$

or (User productivity) $\times$ (Power efficiency).

From such a perspective, human factors and user interfaces have a large impact on system energy efficiency, simply because they determine not only the power consumption for interaction but also user productivity. Most low-power research has focused on reducing power consumption, given a computation or interactive task. However, it is equally, if not more, important to optimize the interaction itself, *i.e.*, reduce the interaction power and improve user productivity.

In this paper, we focus on the impact of human factors and user interfaces on energy efficiency. To the best of our knowledge, this is the first work of this nature. We first present theoretical studies of the minimal energy/power requirements for user interfaces based on human sensory limits, and then take into account the human speed for human-computer interaction. We then investigate the energy efficiency of the state-of-the-art interfaces by characterizing different interfacing technologies available on two commercial handheld computers. Based on the characterization, we offer a comparative study of energy efficiency of these different interfacing methods. We find that speech-based input has a great potential to become the most energy-efficient interfacing method since we can speak at a much higher rate than we can write or type. Such a comparative study offers guidelines for mobile system designers when choosing interfacing technologies.

As the characterization clearly shows, energy requirements of state-of-the-art interfaces are far from the theoretical minimal. In fact, interfacing components, such as the display and speaker subsystems, are among the most power-consuming components. On the other hand, human capacity is essentially limited, and the computer usually spends most of its time waiting for the human user during interaction. Therefore, significant energy is spent in waiting due to power-hungry interfacing components and a slow user, leading to an energy efficiency bottleneck. Such a bottleneck cannot be removed with more sophisticated user interfaces, which usually

*Acknowledgments: This work was supported in part by NSF under Grant No. CCF-0428446 and in part by a Princeton University Honorific Fellowship.

consume even more power. Motivated by memory-cache theory, we show how a low-power interface cache device with much simpler and lower power interfaces can be used to handle simple interactive tasks outsourced from a host computer, and thus improve the battery lifetime of the latter. It saves energy essentially by bringing the interface energy requirements closer to the theoretical minimal without sacrificing user productivity much for the simple outsourced tasks. In this work, we designed and prototyped a wireless wrist-watch as an interface cache device to serve an HP iPAQ handheld computer.

The paper is organized as follows. We discuss limits on energy efficiency imposed by human factors in Section II. We then offer user interface energy characterization and comparative studies in Sections III and IV, respectively. We present the design and experimental results for the wrist-watch as an interface cache device in Section V. We discuss related works in Section VI, and conclude in Section VII. It is worth mentioning that there are many other issues involved in user interface design and evaluation than energy efficiency, such as user acceptance and form factors. In this work, however, we focus on energy efficiency from a computer engineering perspective.

II. Limits due to Human Factors

This section examines how human factors impose limits on energy efficiency with regard to interfacing power and user productivity (speed). It highlights the importance of human factors and interfaces in determining system energy efficiency as compared to computing. It also provides theoretical foundations for improving user interfaces for better energy efficiency.

A. Sensory Perception-based Limits

Landauer [23] showed that the theoretical lower bound for energy consumption of an irreversible logic operation is $kTln2$, where k is the Boltzmann constant and T is the temperature. kT is of the order of $10^{-21}J$ at room temperature. All commercially available computing devices use irreversible logic operations and are hence governed by this bound. On the other hand, the computer has to communicate with its human user through the latter's sensory channels. These channels in fact set the minimal power/energy requirements for the computer output.

1) Visual output: Human vision energy thresholds have been measured in different forms [4] in terms of minimal absolute energy, minimal radiant flux, and just-perceptible luminance. Minimal absolute energy is measured for a very small solid-angle field, *e.g.*, a point source, presented for a very short time ($10^{-3}s$) so that no temporal summation of radiant flux occurs. Minimal radiant flux is measured for a very small solid-angle field lasting for a long time so that temporal summation of radian flux occurs. Just-perceptible luminance is measured for a large-area visual field. These thresholds are used to estimate the energy/power dissipation lower bound for displaying information as follows.

Minimal absolute energy: Let us assume the user's cornea area is A, viewing distance D, and viewing angle Ω. We assume the light irradiance is the same for every point within the viewing angle that is at the same distance from the point source. Let $E_{min}(\lambda)$ denote the minimal light energy reaching the cornea that is detectable by the user for light of wavelength λ. The total energy emitted by the source is thus:

$$E(\lambda) = \frac{\Omega D^2}{A_i} \cdot E_{min}(\lambda) \approx \frac{\Omega D^2}{A} \cdot E_{min}(\lambda)$$

where A_i is the area of the viewing sphere that is incident on the cornea. A_i is approximated as the cornea area A.

Experimental results reported by psychology researchers [4] indicate that E_{min} for light of wavelength $510nm$ is about $2 \cdot 10^{-17} \sim 6 \cdot 10^{-17} J$. Assuming $A = 0.5cm^2$, $D = 0.3m$, and $\Omega = 0.125 \cdot 2\pi\ sr$, we have $E \approx 3 \cdot 10^{-14} \sim 9 \cdot 10^{-14} J$, which is about seven orders of magnitude larger than the energy required for an irreversible logic operation.

Note that the energy limit derived above is for rod vision, which is human vision under extremely low luminance and colorless. Only the cone vision contains color and is normally required for human-computer interaction. The energy threshold for cone vision for $\lambda = 510nm$ is more than 100 times that of rod vision. For users to sense color, the minimal energy would thus be of the order of $10^{-11}J$.

Minimal radiant flux: Let $R_{min}(\lambda)$ denote the minimal radiant flux for light of wavelength λ that humans can sense. For the viewing distance D and viewing angle Ω, the source radiant power is given by:

$$\Phi_{min}(\lambda) = \frac{R_{min}(\lambda) \cdot \Omega D^2}{683 \cdot V(\lambda)}$$

where $V(\lambda)$ is the relative visibility factor and 683 is the spectral efficiency for $\lambda = 550nm$ in $lumen/W$. According to [4], the minimal radiant flux for white light rod vision is about $4 \cdot 10^{-9} lumen/m^2$. Assuming $V(\lambda)$ for white light to be 0.8, we obtain $\Phi_{min} \approx 5 \cdot 10^{-13} W$ under the same assumptions for D and Ω as before.

Just-perceptible luminance: Suppose the just-perceptible luminance for light of wavelength λ is $L_{min}(\lambda)$. Let S denote the area of the display and Ω the viewing angle. The total display radiant power,

$\Phi_{min}(\lambda)$, is then

$$\Phi_{min}(\lambda) = \frac{L_{min}(\lambda) \cdot S \cdot \Omega}{683 \cdot V(\lambda)}$$

For white light, L_{min} has been determined to be $7.5 \cdot 10^{-7} candella/m^2$ [4]. With the same assumptions as above, the minimal radiant power for a $12.1''$ laptop display and white light is about $5 \cdot 10^{-11} W$. For comfortable reading, the luminance level is, however, about $\frac{100}{\pi}$ $candella/m^2$ [4], which requires a radiant power of about $2mW$ for a $12.1''$ display. This minimal radiant power for comfortable reading is about seven orders of magnitude larger than the just-perceptible threshold.

2) Auditory output: Let Ω denote the solid hearing angle and D the distance between ears and the sound source. The minimal sound intensity human beings can hear is about $10^{-12} W/m^2$ for a sound field of relatively long duration ($>300ms$) [12]. Below $300ms$, the threshold sound intensity increases fast as the sound duration decreases [12]. Therefore, we can estimate the minimal energy, E_{min}, for human beings to detect one bit of auditory information to be

$$E_{min} = 10^{-12} \cdot 300 \cdot 10^{-3} \Omega D^2$$

Assuming $\Omega = 0.125\pi$ sr and $D = 0.3m$, we have $E_{min} \approx 10^{-14} J$, which is of the same order of magnitude as the minimal energy required for displaying one bit of visual information. Note that the minimal sound intensity varies for sounds of different frequencies. $10^{-12} W/m^2$ is approximately the just-perceptible intensity of sound at a $1000Hz$ frequency, which belongs to the span of frequencies human beings are most sensitive to. A normal conversation generates a sound level that is about 10^6 times larger than the just-perceptible sound intensity. Therefore, for a user to obtain auditory information from a computing system, the sound intensity should be no less than $10^{-6} W/m^2$. For the values of Ω and D given above, this results in an acoustic energy requirement of about $10^{-8} J$. Moreover, the above thresholds assume no noise (just-perceptible intensity) or relatively low noise (conversational intensity). When ambient noise increases, the output sound level has to increase accordingly, according to Webber's Law [12].

3) Power reduction techniques: Based on the above discussion, we can formulate the power requirement of a visual/auditory output as follows

$$P \propto \frac{\Omega \cdot D^2}{\eta(\lambda) \cdot V(\lambda)} \quad (1)$$

where $\eta(\lambda)$ is the conversion efficiency from electrical power to light/sound radiant power for wavelength λ, and $V(\lambda)$ the relative human sensitivity factor. Most display research efforts have been devoted to improving $\eta(\lambda)$ by adopting new display devices. For organic light-emitting devices (OLEDs), the best $\eta(\lambda)$ so far is $70 lumen/W$ for $\lambda = 550nm$ [10]. This is about 10-fold smaller than the theoretical $683 lumen/W$ upper limit [4].

Reducing the viewing/hearing distance D seems to be the most effective way to reduce output power requirement. Unfortunately, it poses a practical problem for visual output since it requires changes to the way a display is used. Moreover, reducing D may also have an impact on other display parameters such as pixel size and aperture (the ratio of the effective area to display area). A head-mounted display is a successful example where a reduced D is used. However, it is promising only for limited scenarios such as military and virtual reality applications at this moment. Unlike head-mounted displays, their auditory counterparts, earphones, are quite popular. Due to their extremely small D and Ω, earphones are much more power-efficient than loudspeakers, as we will see in Section III.

Moreover, many applications do not need a large viewing/hearing angle. The viewing/hearing angle can be controlled to reduce output power consumption too. Another hint from Equation (1) is that choosing the colors/sounds with a higher human sensitivity, thus higher $V(\lambda)$, will also reduce power. Human vision sensitivities to different colors differ by several orders of magnitude. However, user experience with colors is quite complicated since color contrast and aesthetics also matter.

B. Input/Output Speed

The energy consumption per task depends not only on system power consumption but also on the task duration, or speed. We next characterize input/output speeds for human-computer interaction, which will be used to compare the energy efficiency of different interfacing technologies in Section IV. This subsection draws upon many previous surveys [1].

Speaking/Listening/Reading speeds: 150 words per minute (wpm) is regarded as normal for conversational English for both speaking and listening. When speaking to computers, users tend to be slower at about $100wpm$ [20]. Also, users can listen to compressed speech at about $210wpm$ [29]. Such speaking and listening rates set limits to the energy efficiency of speech-based interfaces, as shown in Section IV. Moreover, when speech-recognition errors have to be corrected, the speaking rate is reduced drastically to as low as $25wpm$ [20]. For reading printed English text, 250 to $300wpm$ is considered typical [5]

Text entry: Text entry on handheld devices is well-known to be much slower than on PCs with a full-size QWERTY keyboard. Table I summarizes results

from the literature about input speeds for popular text entry methods available on commercial handheld computers, such as HP iPAQ and Sharp Zaurus, which are studied in this work. "Typical speed" refers to the raw speed regardless of accuracy while "Corrected speed" refers to real speed when error correction is taken into consideration. Note that handwriting speed is for hand-printing, which serves as an upper bound for the input speed for any handwriting recognition-based text entry. The corrected word per minute ($cwpm$) for handwriting recognition is around 7 [8]. We assume that the error rate is low for hardware mini-keyboard thumbing, *i.e.*, typing with two thumbs, and error correction is fast, as assumed for the virtual keyboard in [8].

TABLE I
TYPICAL TEXT-ENTRY SPEEDS FOR DIFFERENT METHODS

Method	Typical speed (wpm)	Corrected speed ($cwpm$)
Hardware mini-keyboard	23 [33]	22
Virtual keyboard	13 [32]	12 [8]
Handwriting	15 [25]	N/A

Stylus/touch-screen: For GUI-based human-computer interaction, the speed is usually dependent on how fast the user can respond to the GUI. In [37], we characterized the user delays and investigated how they could be predicted for aggressive power management. As we are more interested in typical delays for energy-efficiency evaluation, we assume that a 500 to $1000ms$ user delay is typical for GUI operations on handheld devices.

III. Energy Characterization

The previous section demonstrated that human factors impose limits on both interfacing power consumption and interaction speed. It also offered the theoretical minimum power requirements for interfacing. Although such minimum power requirements are orders of magnitude larger than those for computing, they are still far from the reach of the state of the art user interfaces as we will see in this section.

A. Characterization Setup

Table II provides information on system settings and input methods for the two handheld computers characterized in this work. iPAQ is also equipped with Bluetooth. Note that several different handwriting recognition schemes are available on both computers. The user can input text letter-by-letter using *letter* or *block recognition* on both computers. The user can also input a group of letters using Microsoft *Transcriber* [27] on the iPAQ.

Power measurements: Power measurements are obtained by measuring the voltage drop across a $100m\Omega$ sense resistor in series with the $5V$ power supply cord. The measurement system consists of a Windows XP PC with a GPIB card and an HP Agilent 34401A digital multimeter. A program, developed with Visual C++, runs on the PC and controls the digital multimeter to measure the voltage value. The value is sampled about 200 times per second.

TABLE II
SYSTEM INFORMATION FOR IPAQ AND ZAURUS

	iPAQ	Zaurus
Model	HP iPAQ 4350	Sharp SL5600
SoC	Intel XScale $400MHz$	
Storage	$32MB$ ROM, $64MB$ RAM	$16MB$ ROM, $64MB$ RAM
Display	240×320, 16-bit color	
	Transflective/back light	Reflective/front light
OS	MS Pocket PC 2003	Embedix Plus PDA 2.0 (Linux 2.4.18)
Battery	$1560mAh/3.7V$	$1700mAh/3.7V$
Text entry	Touch-screen with stylus	
	Hardware mini-keyboard (QWERTY)	
	Virtual keyboard (QWERTY)	
	Handwriting recognition	
Image/Video	N/A	CF digital camera
Audio	Integrated mic., speaker & headphone jack	
Speech recog.	Voice Command [28]	N/A

Basic power breakdown: We first characterize the power consumption due to hardware activities initiated by user interaction. We use the power consumption of idle PDAs (in the IDLE mode) with the display off as the baseline, and present the power consumption of additional hardware activities as extra power consumption relative to the baseline. The extra power/energy consumption [36] of an event is obtained through two measurements: one for the system power/energy consumption during the period an event of interest occurs; the other for the system power/energy consumption during the same period when the event does not occur. For example, the extra power consumption of the LCD is obtained by subtracting the system power when the system is idle and the LCD is off from that when the system is idle and the LCD is on. The power characterization results are presented in Fig. 1. In this figure, "BT Trans." refers to Bluetooth transmitting data at 9.6 Kbps; "BT Paging" refers to Bluetooth seeking a connection with another device, and "Comp." refers to measurements when the system is executing a discrete-cosine transform (DCT) application repeatedly.

B. Visual Interfaces

We first examine visual interfaces.

Graphical user interface: In [36], we presented a comprehensive analysis of the system energy consumption required for GUI manipulations. In [37], we

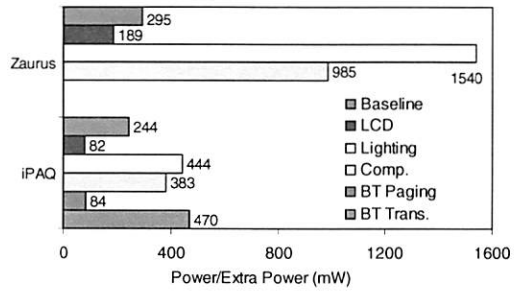


Fig. 1. Baseline power and extra hardware power consumption

showed, however, that most of the system energy is consumed when the system waits for the next user input. If we ignore the extra energy consumed by the system to generate a GUI response, GUI manipulation-based interfaces basically consume energy through a static display and an idle system. As pointed out in [36], the most effective system energy reduction strategy is to improve user productivity so that more tasks can be accomplished given the same battery lifetime. In Section IV, the energy efficiency of GUIs is compared with other interfacing technologies based on the length of the corresponding GUI operation.

Visual input: Gesture recognition and lip-reading have been proposed as possible techniques for multimodal human-computer interaction. Both require video or image input. We used a CF digital camera card on Zaurus to obtain its power cost. When the camera is turned on with a 480×320 resolution and faces a static object, the system consumes about $1.35W$. When the object moves, the power consumption increases slightly to about $1.36W$. This is close to the power consumption when the user is preparing for a shot, *e.g.*, adjusting the focus and view. Also, it takes about $0.33J$ to capture a 480×320 picture. Since a user usually takes more than a few seconds to prepare a shot, it is obvious that it is more important to reduce the user's preparation time and system power consumption during that time than reduce these for actually capturing the picture.

C. Auditory Interfaces

We next examine the auditory interfaces available on iPAQ and Zaurus.

1) Direct recording and playback: An auditory signal can be directly recorded and played back for interfacing purposes. Direct recording is often used for note-taking and direct playback for short sound responses from the computers such as warnings and notifications. If there are too many sound responses to be feasible for direct playback, speech synthesis is required.

Direct recording: iPAQ provides a hardware button to start recording ($11KHz$ 16-bit Mono), which is very useful for audio note-taking. The recording consumes $525mW$. The extra power consumption is thus $199mW$. Zaurus draws about $198mW$ extra power consumption when recording ($16KHz$ 16-bit Mono).

Direct playback: A WAV sound clip ($32KHz$ 16-bit mono) was played on both iPAQ and Zaurus. To separate the power consumption of the speaker subsystem, the clip was played at different volumes. Table III shows the power consumption under various scenarios. "Half" volume assumes that the volume controller is set at the half mark on each system. In this scenario, the clip is not comfortably enjoyable on either system, even in a quiet office environment, if the system is about two feet from the user head. All the system power numbers include that consumed by the LCD.

The extra power consumption of the speaker subsystem is obtained by comparing the system power consumption before and after the system is muted. This has a significant impact on system power efficiency if the auditory output is used. Notably, using an earphone instead of the built-in loudspeaker reduces power by more than $300mW$ and $410mW$ for iPAQ and Zaurus, respectively. The data also indicate that using a simpler audio format (WAV as opposed to MP3) reduces power consumption at the cost of increasing the storage requirement. Since the extra power consumption of the speaker subsystem for playing MP3 is similar to that for playing WAV, the power consumption for playing MP3 at different volumes is not presented.

2) Speech recognition and synthesis: We next examine the Microsoft Voice Command [28] on iPAQ to obtain its power for a speech recognition-based interface. Voice Command is similar to the MiPad Tap & Talk system [22] except that synthesized speech is used as feedback to the user. Not having a detailed knowledge of its implementation, we adopted a black-box approach. We recorded both the power trace and the audio input/output and then aligned them to divide the power trace into meaningful segments. We fed different inputs to Voice Command to elicit certain behaviors from it.

Speech acquisition without speech being detected: We first evaluated Voice Command under no sound. Hence, the speech detection module does not detect any speech. A reasonable speech recognition implementation will discard most of the acquired speech without performing feature extraction under this scenario. Therefore, the power consumption can be attributed to the microphone subsystem and speech detection module. From the power trace, we observed that Voice Command calls the speech detection module about every $250ms$. Each call contributes to a peak in the power trace, leading to an average extra power consumption of $126mW$.

Speech acquisition with speech being detected: We next evaluated Voice Command when fed with irrel-

TABLE III

POWER CONSUMPTION FOR DIFFERENT AUDITORY OUTPUTS

Format	Volume	iPAQ (mW)			Zaurus (mW)		
		System	Extra	Speaker	System	Extra	Speaker
WAV	Max.	747	420	367	1,030	546	422
	Half	552	232	172	637	153	29
	Muted	380	53	0	608	124	0
	Earphone Max.	445	118	65	619	135	11
MP3	Earphone Max.	476	149	N/A	632	148	N/A

evant utterances, which are detected as speech but not recognized. The power trace generated was very similar to the one when there was no speech except that the peaks became wider when the input utterances became more continuous. These wider peaks can be attributed to feature extraction performed immediately after speech is detected and recognition decoding after a certain amount of speech is detected. The typical power consumption for processing a continuous irrelevant utterance is about $780mW$. Interestingly, if the utterance is relevant or recognizable, the average power consumption is actually much lower. For all the traces we obtained with a valid command, the power consumption is usually about $680mW$ in this case. The higher power consumption with irrelevant utterances may be introduced by a larger search space. For valid utterances, the search space can be significantly pruned because some very promising search paths can be identified early.

Speech synthesis: We recorded the power trace for iPAQ when it synthesized the speech output for speech recognition. The extra power consumption for the speaker subsystem at maximum volume is $181mW$, which is significantly smaller than that shown in Table III. This is due to the fact that the sound clip used for generating the table is a continuous flow of music while the synthesized speech output only uses the speaker subsystem intermittently, leading to a much lower duty-cycle. The non-speaker subsystem power for speech synthesis is about $75mW$. Compared to the $383mW$ extra power required for performing DCT (see Fig. 1), such a speech synthesis is not computationally demanding on iPAQ at all.

It is worth noting that for many voice commands, the display need not be on. This means that 82 to $526mW$ $(82 + 444)$ power reduction is possible (see Fig. 1). As we will see in Section IV, speech recognition-based interfaces are more energy-efficient in many scenarios only if the display is turned off compared to several other interfacing technologies.

D. Manual Input Techniques

We next characterize the extra energy consumption for various manual input techniques for text entry. For letter-based input, such as letter recognition and virtual keyboard, we examine the extra energy consumption for inputting a letter; for word-based input, such as *Transcriber*, we examine the extra energy consumption for inputting words of different lengths. Table IV presents the extra energy consumption for inputting a letter. Fig. 2 presents the extra energy consumption for inputting words of different lengths using *Transcriber* on iPAQ. The energy consumption per letter increases slightly as the word becomes longer due to a larger recognition effort.

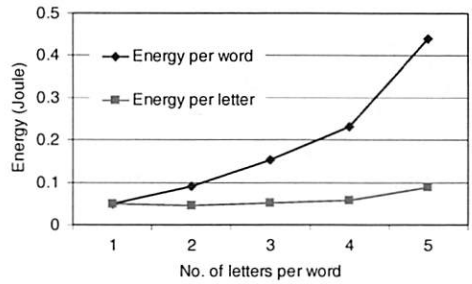


Fig. 2. Extra energy per word/letter for *Transcriber*

The above text-entry methods consume energy through touch-screen usage and related CPU activities. However, the energy thus consumed is insignificant compared to that consumed by the LCD, which needs to be on during text entry. Therefore, the energy cost per letter is not the only indicator of the energy efficiency of a text entry method. What matters more is the entry speed, as we will see in Section IV.

TABLE IV

EXTRA ENERGY CONSUMPTION FOR INPUTTING A LETTER

Input method	Extra energy (mJ)	
	iPAQ	Zaurus
Hardware keyboard	~30	~50
Virtual keyboard	~10	~80
Letter recognition	~30	~330

IV. A Comparative Study

Based on the discussion of interaction speeds and energy characterization presented in Sections II and III, we next compare the energy efficiency of different user interfaces. As speech-based interfaces are gaining ground, we use such an interface as the baseline.

A. Output

We first examine the energy efficiency for presenting language-based information through speech or text.

When the information to be presented is long enough, the reading/speaking rate determines the duration of presentation. Let R_{spk} denote a comfortable speaking rate and R_{rd} a comfortable reading rate in wpm. Let P_{txt} denote the system power consumption for presenting text. For simplicity, we assume that P_{txt} is roughly constant for presenting any text. We ignore the energy consumed to render the GUI for the text. The computer is basically idle after the text is presented on the display. On the contrary, the computer has to be active when the text is spoken back to the user. Let P_{spk} denote the corresponding system power consumption.

The ratio of energy consumption for text and speech outputs is therefore

$$r_{output} = \frac{R_{spk}}{R_{rd}} \cdot \frac{P_{txt}}{P_{spk}}$$

The following techniques can impact r_{output}: P_{spk} can be changed drastically by turning the display on or off or by using an earphone instead of the loudspeaker; P_{txt} can be reduced by employing aggressive power management [2], [37]. Fig. 3 gives different values of r_{output} for iPAQ and Zaurus under some possible scenarios based on data presented in Section III. We assume $R_{rd} = 250wpm$ and $R_{spk} = 150wpm$. "Light" indicates that the back light or front light is on and "PM" or "NPM" refers to whether aggressive power management [2], [37] is employed or not. The X-axis denotes whether the display, together with lighting for "Light," is on or off and whether the built-in loudspeaker or earphone is used for speech output. For Zaurus, the direct playback power consumption is used as P_{spk}. Note that the speech output is more energy-efficient if and only if the ratio is greater than 1.

For iPAQ, when the back light is on for night-time text reading, a synthesized speech output through an earphone with the display off would be more energy-efficient than a text output. For Zaurus, a speech output consistently consumes more energy for day-time usage when the front light is not needed. It is more energy-efficient only when the display does not need to be on. Its advantage primarily comes from the fact that the speech output does not mandate that the power-hungry display be on. On the other hand, it consumes two to three times more energy if the loudspeaker is used and the display is left on. *The key to improving energy efficiency for a speech output is therefore to turn off the display and adopt a low-power audio delivery method other than a loudspeaker.*

When the information is very short, such as short messages and notifications, the presentation duration is not primarily determined by the reading/speaking rate but other time overheads for eye/hand movements and distraction. Therefore, speech and audio delivery

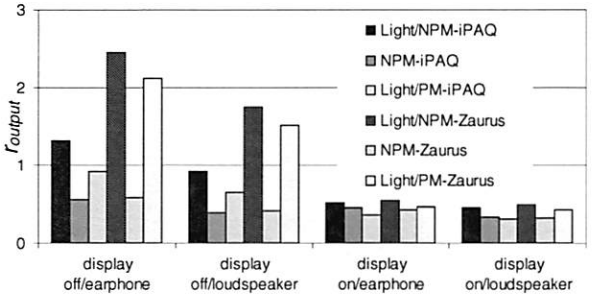


Fig. 3. Ratio of system energy consumptions for text output over speech output under different scenarios

can be very energy-efficient [11], [31] since it is not visually intrusive and persistent. Such short messages and notifications, if delivered as GUI presentations, could interrupt user's ongoing work and require user action to respond, *e.g.*, to close the popup message box, leading to a larger energy overhead.

B. Input

Next, we compare the energy efficiency of different input methods. There are two types of input, namely, text and control.

Text entry: In Section III-D, we derived the extra energy consumption for inputting a letter under different text-entry methods. As pointed out in Section II-B, the corresponding input speeds vary a lot. Let R_{entry} denote the typical input speed in wpm. Let e denote the extra energy consumed for inputting one letter using the method characterized in Section III-D and P_{idle} denote the system idle-time power consumption. We assume an average word requires six letter inputs [5], including a space. On the other hand, let R_{spk} denote a comfortable speaking rate for recognition-based input and P_{recog} the system power consumption during speech recognition. If we ignore the energy consumed during the delay between the end of speech and the end of speech recognition, the ratio of the energy consumptions for manual text entries and speech-based text entries is given by

$$r_{input} = \frac{\frac{P_{idle}}{R_{entry}} \cdot 60 + e \cdot 6}{\frac{P_{recog}}{R_{spk}} \cdot 60} = \frac{R_{spk} \cdot P_{idle}}{R_{entry} \cdot P_{recog}} + \frac{e \cdot R_{spk}}{10 \cdot P_{recog}}$$

Obviously, the energy efficiency of an input method is primarily determined by its input speed.

Although Voice Command is not intended for text entry, we assume speech recognition-based text entry would have similar power characteristics and therefore use the power consumed by the Voice Command recognition process (P_{recog}). Fig. 4 plots the r_{input} for the hardware mini-keyboard (HW MKB), virtual keyboard (VKB), and letter recognition (Letter Recog.) using data from Sections II and III. For each method, four cases are shown. "ideal" refers to typical input speed

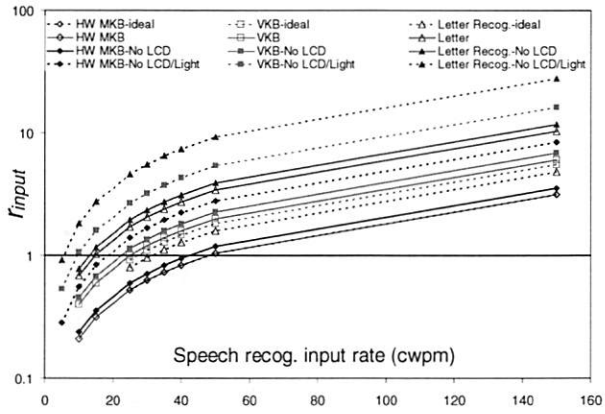


Fig. 4. Ratio of system energy consumption for different text-entry methods over speech-based text entry

without considering error correction; "No LCD" refers to comparisons to speech recognition with the display off; "No LCD/Light" refers to night-time usage with the back light on as compared to speech recognition with the display off. Except for "ideal," input speeds are expressed in $cwpm$. The break-even line with r_{input} equal to 1 is also shown. For any point above this line, the speech recognition-based input is more energy-efficient. From Fig. 4, the potential energy advantages for speech recognition-based text input are obvious since speech is potentially much faster than any other text input method. However, a recognition-based input method usually incurs much higher input errors, leading to a much lower speed in $cwpm$. For example, the speed of handwriting recognition is about half the speed of handwriting. Studies in [8] have shown that speech recognition speeds of $17cwpm$ are already available and may reach 45 to $50cwpm$ in the near future. If the power consumption for correcting errors in speech recognition is about the same as the power consumption during recognition, Fig. 4 shows that speech recognition is already more energy-efficient than letter recognition and also the virtual keyboard for night-time usage if the speech recognition-based interface does not require the display to be on. Moreover, when a speed of 45 to $50cwpm$ is achieved by the speech recognition-based interface, it will be more energy-efficient than most text-entry methods, even the hardware mini-keyboard.

Command and control: Error correction drastically decreases the input speed for speech recognition-based text entry, leading to a much lower energy efficiency. For command/control applications such as Voice Command, however, errors can be corrected much faster, $e.g.$, by reissuing the command. Moreover, for such applications, the recognition accuracy is usually much higher. This leads to a higher throughput and thus a higher energy efficiency. For a command/control task, let us assume it may take M stylus taps or it may

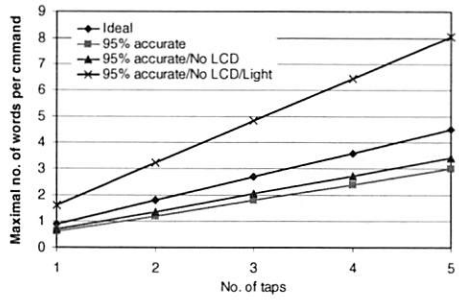


Fig. 5. The maximal number of words per command for better energy efficiency

take a W-word voice command. Let N denote the speaking rate in $cwpm$. Based on the traces collected for PDA usage [37], we assume each stylus tap is accompanied by a $750ms$ user delay, which is mostly an underestimation for typical menu selections on iPAQ. Moreover, we assume that the energy consumed by the GUI response can be ignored compared to that consumed during the user delay. Therefore, r_{cc}, which represents the ratio of the energy consumptions by GUI-based and speech-based command/control, is given by:

$$r_{cc} = \frac{P_{idle}}{P_{recog}} \cdot \frac{M \cdot N \cdot 0.75}{60 \cdot W}$$

Obviously, the shorter a voice command, the more energy-efficient it is. Fig. 5 shows the maximal number of words per command required so that speech-based command/control is more energy-efficient than GUI operations with different numbers of taps under various scenarios. 100% accurate speech recognition with $N = 150$ is used to draw the "ideal" line. Note that $150wpm$ is regarded as the conversational English speaking rate. In other cases, 95% speech recognition accuracy is used with $N = 100$, assuming 10 times more energy/time required to correct an error compared to speech recognition. Such an assumption is pessimistic since most errors can be corrected by simply reissuing the command. "No LCD" and "No LCD/Light" have the same meaning as in Fig. 4.

Fig. 5 shows that *a one-word voice command is more energy-efficient than GUI operations with two or more taps*. If the display can be turned off for speech-based command/control, its advantage is higher.

Taking notes: Speech and handwriting recognition-based text entries are mostly hindered by their low accuracy and high cost for correcting errors. However, if text transcription is not needed in real-time, $e.g.$, when using audio recording or handwriting to take a note, it is most energy-efficient to use speech since speaking is much faster than any other input method. However, if the note has to be retrieved in the format that it was recorded before recharging, there is a tradeoff between the energy consumptions for taking a note and

for retrieving it, especially when it has to be retrieved multiple times.

C. Observations

The energy characterization and comparison presented so far have provided concrete and practical justifications for defining system energy efficiency as $\frac{\text{User productivity}}{\text{Average power consumption}}$. Based on such a characterization and comparison, we can make the following observations for improving energy efficiency.

Speed matters: The faster a task is accomplished and the higher the user productivity, the more energy-efficient the system usually is. From this perspective, *interface designers share a significant responsibility for designing an energy-efficient system.* In most cases, improving user productivity may incur average power consumption increase. As long as the productivity improvement percentage is larger than average power increase percentage, the system energy efficiency is improved.

In terms of the specific interfacing methods, speech-based input stands out since speech is inherently much faster than other input methods. For recognition-based input, such as handwriting and speech recognition, accuracy is important due to the high cost of correcting errors. Thus, accuracy is also important for system energy efficiency.

Display matters: The energy efficiency for a display-based interface suffers a lot since its average power consumption includes that of the display, which is large. Touchscreen/stylus-based interaction basically integrates the input hardware with the output hardware, leading to a high power consumption even for making an input, especially for night-time usage. When the power-hungry display has to be on with a slow input rate, *e.g.*, for all the manual text-entry methods, energy efficiency is drastically reduced. Speech-based interfaces again may enjoy an energy efficiency advantage since their display usage can be carefully avoided.

The power consumption landscape, however, is likely to change in a few years due to progress in new display technologies. OLEDs [10] promise high-quality low-power flexible displays for mobile computers. More importantly, bistable display technologies [21], [35] will reduce the static power consumption to nearly zero. This will significantly reduce the energy that a system spends in waiting for user inputs.

Audio matters: Surprisingly, our energy characterization results showed that the speaker subsystem is also power-hungry, drawing as much as $367mW$ and $422mW$ of power for iPAQ and Zaurus, respectively. This power consumption can be drastically reduced by using earphones instead of loudspeakers. However, the wires connecting the earphones may impact other usage issues. Since Bluetooth consumes more than $470mW$ extra power when actively transmitting data (see "BT Trans." in Fig. 1), a Bluetooth headset is unlikely to reduce the audio delivery power consumption. Therefore, *for better exploiting the speed of speech-based interfaces, low-power wireless voice stream delivery between the user and computer is critical.*

Fig. 6. The prototype of a watch as the interface cache for iPAQ

V. Interface Cache

In the previous sections, we investigated how human factors limit energy efficiency for handheld computer interfaces, characterized different interfacing methods, and presented a comparative study for them. Since the computer responds to the user much faster than the latter responds to the former, an increasingly powerful computer, in terms of display and processor, has to spend most of its time and energy waiting for a consistently slow user. Such a speed mismatch is essentially imposed by human capacity and is growing, leading to a bottleneck in energy efficiency even for a system with a perfect user interface. The cache solution to address the speed gap between the processor and memory in conventional architectures [15] motivated us to design a low-power wireless device, to which the host computer can outsource simple interactive tasks. As an interfacing solution to alleviate the energy efficiency bottleneck due to a slow user, such a device functions like a cache for more expensive interfaces on the host, reducing interfacing energy requirements without sacrificing user productivity much. Its design and prototype are discussed next.

A. Wireless Cache for Interfacing

The cache device we designed and prototyped takes the form of a wrist-watch that communicates with a handheld computer wirelessly. The watch prototype and its host, the iPAQ used in the characterization, are shown in Fig. 6.

As an interface cache, the watch provides the host computer with a limited display. It provides two different services: active and passive. The watch stays

connected with the host and waits for user input for the active service. The user input can be echoed on the watch in real-time. For passive service, the watch communicates with the host from time to time to receive data. The connection can be closed after data transmission but the user can still access the data cached on the watch. The watch provides its services through a simple application-layer protocol, called the synchronization protocol. It is up to the host computer to determine what and how to display and how the connection is managed using the protocol. The watch simply follows instructions from the host computer as a slave.

Hardware components: The watch consists of three major components: a Microchip PIC16LF88 microcontroller, a 2×8 monochrome character LCD, and a Bluetooth-RS232 adapter (Promi-ESD class II) from Initium [17]. One MAX604 linear regulator is also used. The system is powered by a 3.6V supply with three $800mAh$ rechargeable AAA batteries. The microcontroller is run at a $10MHz$ clock frequency, drawing a current of less than $0.6mA$. It drives the LCD module directly but controls the Bluetooth adapter through a $9.6Kbps$ serial port. The LCD module draws a current of about $1mA$. There is no lighting for the LCD module in the prototype, a limitation of the current implementation that can be easily alleviated. Therefore, we assume that the LCD module can be used at night time in the following discussion.

The Bluetooth-RS232 adapter implements the simplest Bluetooth application profile, the Serial Port Profile (SPP). Two devices with Bluetooth SPP can communicate in the same way they would with an RS232 serial port connection. The Bluetooth adapter has a number of operational modes. When it is not connected or seeking connection, it is in the STANDBY mode, drawing about $6mA$ current, but can be turned off for more energy savings. In the following discussion, we use the STANDBY mode to refer to both situations. When the Bluetooth adapter is seeking connection through a Page-Scan session, it is in the PENDING mode, drawing about $19mA$ current. In the PENDING mode, it does Page-Scan for T_{ps} ms every T_{pc} ms. T_{ps} must be a multiple of a $625\mu s$ slot. Both T_{ps} and T_{pc} can be configured through commands from the RS232 serial port, leading to different average power consumption. When the Bluetooth adapter is connected, it is in the ACTIVE mode, drawing about $9mA$ current for no data transmission and about $28mA$ during active data transmission at $9.6Kbps$.

Communication design: Since the data for the passive service are not time-critical, the host buffers data for a certain period of time and a connection to the host is required only from time to time. The communication between the watch and its host computer is not data-intensive and communication occurs only sporadically. Therefore, energy consumption due to data communication is very small compared to that required to establish a connection. However, based on our energy characterization data presented in Section V-B, it is more energy-efficient to disconnect and then reestablish a connection in a cooperative fashion when the connection is required more than 30 seconds later. By "cooperative," we mean that both the watch and the host enter the PENDING mode at the same time.

When the host is connected to the watch, it schedules the next communication with the watch according to prior-history-based prediction. It then determines whether the current connection should be maintained or closed based on when the connection will be required the next time. If the connection needs to be closed, the host notifies the watch when it will seek connection next time. After receiving such a notification, the watch shuts down its Bluetooth adapter and forces it back into the PENDING mode when the specified time has elapsed. This ensures that a connection is established in a cooperative fashion, and keeps the watch and the host synchronized with a relatively low energy overhead, as we will see in Section V-B. If the watch loses synchronization with the host, they enter the PENDING mode to re-synchronize.

Software design: The software on PIC is developed using PicBasic Pro. In the main loop, PIC reads its hardware UART and interprets the data according to the synchronization protocol. Following instructions from the host or user, PIC can send AT commands to change modes for the Bluetooth adapter.

Software on the iPAQ, called *watch manager*, was developed using Embedded Visual C++ and built upon the BTAccess library [3]. The watch manager functions like a device driver. On the one hand, it implements the synchronization protocol. On the other hand, it collects information from different application software, such as Outlook, according to the user configuration. The information is buffered in the watch manager and then sent to the watch when it connects to the host. Each time it is connected to the watch, the manager schedules the next connection and notifies the watch through the synchronization protocol.

Interface design: For this prototype, we used very simple interfaces to support the targeted services: a 2×8 monochrome LCD and three tact buttons, Buttons 1, 2, 3. The user can change the watch service mode by clicking Button 3. When the watch is in the passive service mode, a Button 3 click puts the Bluetooth adapter into the PENDING mode unless the connection with the host is established. When the watch is in the active service mode, a Button 3 click simply closes the connection and brings the Bluetooth adapter back to

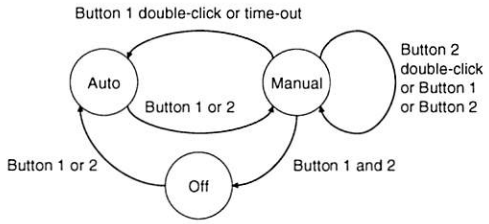


Fig. 7. State-machine description of the watch interface

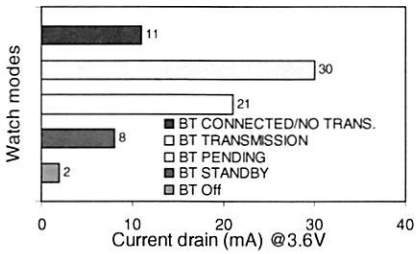


Fig. 8. Power consumption for the watch in different modes

the STANDBY mode to wait for the next scheduled connection.

In the active service mode, a Button 1 click clears the display and sends a negative confirmation back to the host, while a Button 2 click simply sends a positive confirmation. In the passive service mode, the user can use Buttons 1 and 2 to browse the text messages cached in the watch. The interface is better represented as a finite-state machine, as shown in Fig. 7. In the $Auto$ state, the watch displays valid message entries in its message cache by rolling the text messages through the first line of the LCD. Each message is rolled according to its meta-data, which specify its priority in terms of how many times it has to be repeated with each display cycle. In the $Manual$ state, the user can use Buttons 1 and 2 to browse valid entries. Clicking Button 1 induces a skip to the next valid entry whereas clicking Button 2 rolls the current message on the LCD by one letter. Double-clicking Button 2 marks the entry currently on the LCD as invalid and confirmed, which is conveyed to the host next time the watch gets connected to the host.

B. Evaluation

We evaluated the prototype watch device as the wireless cache for iPAQ in order to see whether or by how much it would improve the battery lifetime of iPAQ. Instead of using user studies and subjective metrics, we focused on evaluating the design with related objective metrics. We first present the energy characterization results and then an analysis of energy-efficiency improvement.

Since the non-Bluetooth components on the watch are not power-managed, they draw about $2mA$ current all the time. Fig. 8 presents the power consumption for the watch in terms of current consumption at 3.6V when the Bluetooth adapter is in different modes. Fig. 9 shows how the watch battery lifetime changes when the average communication interval changes. Most of the

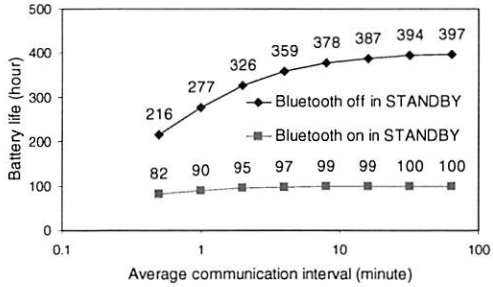
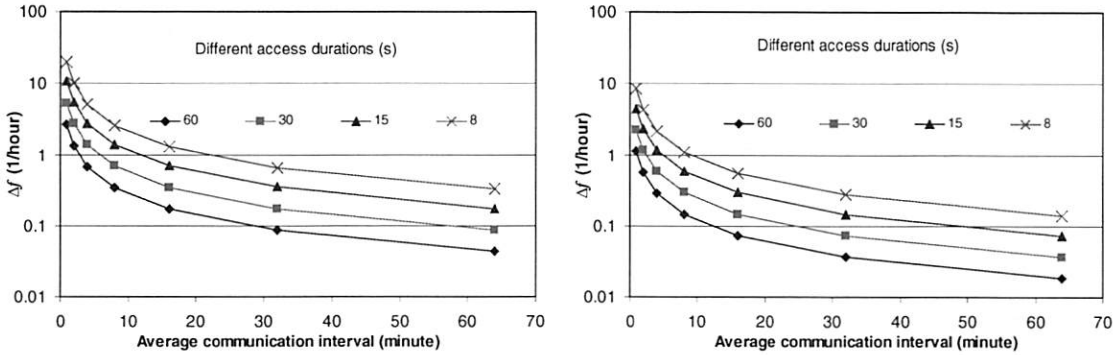


Fig. 9. Watch battery lifetime for different average communication intervals

time, the watch stays disconnected from iPAQ and there is no energy cost for Bluetooth. Assuming a typical $1KB$ data exchange each time a connection is made, the time for data exchange is about 0.118 second. The corresponding energy cost for iPAQ is $84.2mJ$ (based on a power consumption of $470+244 = 714mW$ from Fig. 1). Let T_p denote the time it takes the watch and iPAQ to establish a connection cooperatively and T_s denote the average interval between two communication events. Note that when iPAQ is in the PENDING mode doing Paging, the power consumption is $P_{host} = (84+244) = 328mW$ (see Fig. 1). On the other hand, we assume that if the watch is not used, the user has to access iPAQ directly at a frequency f with average duration of T_{access} ($f < 1/T_{access}$). The average power consumption, P_h, of iPAQ during usage, is about $(244+82+444) = 770mW$ and $(244+82) = 326mW$ with and without the back light, respectively. Let Δf denote the reduction in the iPAQ access frequency f due to the use of the watch. Therefore, using the watch improves the energy efficiency of iPAQ if

$$\frac{84.2 + P_{host} \cdot T_p}{T_s + 0.118 + T_{access}} < P_h \cdot \Delta f \cdot T_{access} \qquad (2)$$

where the left hand side gives the extra power consumption in iPAQ due to Bluetooth activities and the right hand side gives the power reduction due to reduced iPAQ accesses. Fig. 10 plots the minimal frequency reduction in terms of number of accesses per hour for the cache device to improve the iPAQ energy efficiency with different average communication intervals (T_s) and average access durations (T_{access}). Based on our measurements, $T_p = 2.5s$ is used. It presents the results for day-time (without the back light) and night-time (with the back light) access. Different lines represent different average access durations from 8 to 60 seconds. The figure clearly shows how many accesses per hour have to be outsourced to the cache device to improve the host computer energy efficiency. For day-time usage,



(a) Day-time access without the back light

(b) Night-time access with the back light

Fig. 10. Minimal frequency reduction for improving energy efficiency

if the cache communicates with the host every 30 minutes on an average, the host energy efficiency will be improved even if only two 30-second accesses or three 15-second accesses can be outsourced in a day. For night-time usage, even half the number of such outsourcings will still improve energy efficiency.

C. Design Issues

We have shown how the watch can improve the energy efficiency for the host with objective measures. Such a watch is one example of wireless interface cache devices. In the following discussion, we highlight the important issues involved in the design of such devices.

Wireless communication: In terms of form factor, wireless communication between the host and cache devices is almost mandatory. There are several wireless personal-area network (PAN) technologies intended for different data rate and application scenarios. We used Bluetooth in the current watch prototype because its availability on iPAQ and in the market facilitates prototyping. Unfortunately, Bluetooth, especially the iPAQ Bluetooth, imposes a high energy overhead for iPAQ to establish a connection with the watch. This issue can be addressed by implementing the iPAQ Bluetooth in a separate hardware system. In fact, IEEE 802.15.4 or customized radio modules with a lower power consumption, shorter connection establishment time or connectionless communication may be employed to replace Bluetooth since the required data rate is not high. This, however, requires extra hardware to be added to iPAQ.

Tasks for outsourcing: User studies will be critical for determining which tasks should be outsourced to obtain the best tradeoff between the energy efficiency of the host computer and complexity/cost of the cache device. First, instead of determining which tasks should be outsourced, the cache device designer should instead determine which input/output services the cache device should provide. For example, the watch exports its display services through the synchronization protocol and any iPAQ application can utilize such services by specifying what to display. This gives application developers and users most flexibility. Second, even if a task itself suffers productivity degradation after being outsourced to a cache device, system energy efficiency and overall productivity may still be improved. For example, browsing a text message from the watch will be slower than reading it directly from the iPAQ display. However, if the overhead for the user to take out the iPAQ and power it on/off is considered, obtaining information from the watch may even be faster. As another example, laptop usage has been shown to decrease when a BlackBerry smartphone is used [13] because a user can better utilize downtime for productivity, *e.g.*, reading/writing emails with the smartphone while waiting in a line.

Battery partition: A simple question for our interface cache proposal would be: what if we just give the extra battery capacity of the cache device to the host. In the prototype, such an extra battery capacity will give iPAQ an extra operating time of about two hours. The rationale behind a cache device is that we can achieve a longer system operating time by giving some battery capacity to a low-power interface cache device. Therefore, when designing an interface cache device, we must consider the usage patterns of both the cache and host devices and user's expectations of their battery lifetimes to achieve the best system operating time. Again, user studies will be extremely important.

D. Related Devices

Similar wrist-worn devices have been designed including the IBM Linux Watch [16] and Microsoft SPOT watch [26]. They were intended to be a self-contained computer system. If viewed as cache devices for interfacing, they lie on the other extreme of the spectrum, embracing a feature-rich and power-hungry design. Indeed, the author of [14] blamed the high price and short battery lifetime for the lackluster market reception of the Microsoft SPOT watch. They differ drastically from our design of the interface cache device,

which emphasizes a low-power minimalist approach. The TiltType wrist device demonstrated in [9] displays text messages from the computer using an XML-like protocol. With a similar minimalist approach, it was investigated primarily as an input device based on TiltType. None of these devices was proposed to improve the energy efficiency of the host computer.

Related to the philosophy of using a low-power interface cache device, Intel's personal server project envisions that the personal server, a handheld computer-like device, utilizes wall-powered displays in the environment [34]. Since it can be much more energy-efficient to transmit user interface specifications wirelessly than rendering and presenting the user interfaces, such a parasitic mechanism can be another user interface solution to overcome the energy efficiency bottleneck due to a slow user.

VI. Related Work

Low-power research so far has offered many solutions centered around the computer without much regard to the way it interacts with the user, except a statistical request model. Only recently, works have been reported for interface devices such as displays [6], [7], [18], [30], [36]. Also, human factors and user interfaces are recognized as having a significant impact on system energy efficiency. Lorch and Smith [24] pointed out the importance of using user interface events for dynamic voltage scaling. In [36], we characterized the energy consumption of different graphical user interface (GUI) features on handheld computers. We pointed out the importance of idle time and user productivity in system energy efficiency. In [37], we also proposed techniques to aggressively power-manage the system during idle periods based on user-delay predictions. In [2], the authors implemented an aggressive OS-based power management scheme for exploiting idle periods on an Itsy system. Similar techniques were also implemented on the IBM Linux watch [19].

VII. Conclusions

In this work, we presented a comprehensive treatment of energy efficiency considerations for handheld computer interfaces. We showed how human capacities impose limits on system energy efficiency and characterized the energy cost for interfaces on two commercial handheld computers. Based on energy characterization, we presented a comparative study of different interfacing technologies. Specifically, we found that speech-based input has a large potential for outperforming other input methods due to the fact that a human can speak much faster than write or type. On the other hand, a speech-based output suffers from high power consumption required for audio delivery without enjoying a significant speed advantage over text-based output. We also pointed out that the speed mismatch between users and computers and power-hungry interfacing components introduce a bottleneck in system energy efficiency. To solve this problem, we proposed a low-power low-cost device to which a host computer outsources simple, yet frequent, tasks. Such a device, serving a similar goal as the cache does for memory, is called the interface cache. We designed and prototyped a Bluetooth wrist-watch as the interface cache for an HP iPAQ handheld computer. While most other digital watches were designed as complete stand-alone computer systems, our watch is designed purely to serve the host computer. The system functionality is carefully partitioned so that only minimal functionality is placed on the watch. Our experiments and analysis show that such an interface cache will be able to improve the energy efficiency of its host computer significantly.

Acknowledgments

The first version of the Bluetooth wrist-watch was designed/prototyped by Lin Zhong with Mike Sinclair while at Microsoft Research as a summer research intern. The watch was designed to form a PAN of wireless interfacing devices to provide pervasive interfacing for a handheld device like iPAQ. The authors would like to thank all the anonymous reviewers and the paper shepherd, Dr. Mark Corner, for their suggestions that significantly improved the paper.

References

[1] BAILEY, R. W. *Human Performance Engineering: Design High Quality Professional User Interfaces for Computer Products, Applications and Systems*, 3rd ed. Prentice Hall PTR, Upper Saddle River, NJ, 1996.

[2] BRAKMO, L. S., WALLACH, D. A., AND VIREDAZ, M. A. μSleep: A technique for reducing energy consumption in handheld devices. In *Proc. Int. Conf. Mobile Systems, Applications, and Services* (June 2004), pp. 12–22.

[3] BTACCESS LIBRARY. *http://www.high-point.com.*

[4] BUSER, P., AND IMBERT, M. *Vision*. The MIT Press, Cambridge, MA, 1992.

[5] CARVER, R. P. *Reading Rate: A Review of Research and Theory*. Academic Press, Inc., San Diego, CA, 1990.

[6] CHENG, W.-C., AND PEDRAM, M. Power minimization in a backlit TFT-LCD display by concurrent brightness and contrast scaling. *IEEE Trans. Consumer Electronics 50*, 1 (Feb. 2004), 25–32.

[7] CHOI, I., SHIM, H., AND CHANG, N. Low-power color TFT LCD display for handheld embedded

systems. In *Proc. Int. Symp. Low Power Electronics & Design* (Aug. 2002), pp. 112–117.

[8] COMMARFORD, P. M., AND LEWIS, J. R. Models of throughput rates for dictation and voice spelling for handheld devices. *Int. J. Speech Technology 7*, 1 (2004), 69–79.

[9] FISHKIN, K. P., PARTRIDGE, K., AND CHATTERJEE, S. User interface components for lightweight WPANs. *IEEE Pervasive Computing Magazine*, 4 (Oct.-Dec. 2002).

[10] FORREST, S. The roadmap to high efficiency organic light emitting devices. *Organic Electronics 4*, 2-3 (Sept. 2003), 45–48.

[11] GAVER, W. W. Auditory icons: Using sound in computer interfaces. *Human-Computer Interaction 2* (1986), 167–177.

[12] GELFAND, S. A. *Hearing: An Introduction to Psychological and Physiological Acoustics*, 3rd ed. Marcel Dekker, Inc, New York, NY, 1998.

[13] GOLDMAN SACHS GLOBAL EQUITY RESEARCH. *Goldman Sachs Mobile Device Usage Study*. 2001.

[14] GOLDSTEIN, H. A dog named SPOT. *IEEE Spectrum 41*, 1 (Jan. 2004), 72–73.

[15] HENNESSY, J. L., PATTERSON, D. A., AND GOLDBERG, D. *Computer Architecture: A Quantitative Approach*, 3rd ed. Morgan Kaufmann, 2002.

[16] IBM LINUX WATCH. *http://www.research.ibm.com/WearableComputing*.

[17] INITIUM PROMI-ESD CLASS II. *http://www.initium.co.kr/english/promi-esd.html*.

[18] IYER, S., LUO, L., MAYO, R., AND RANGANATHAN, P. Energy-adaptive display system designs for future mobile environments. In *Proc. Int. Conf. Mobile Systems, Applications, & Services* (May 2003), pp. 245–258.

[19] KAMIJOH, N., INOUE, T., OLSEN, C. M., RAGHUNATH, M. T., AND NARAYANASWAMI, C. Energy trade-offs in the IBM wristwatch computer. In *Proc. Int. Symp. Wearable Computers* (Oct. 2001), pp. 133–140.

[20] KARAT, C.-M., HALVERSON, C., HORN, D., AND KARAT, J. Patterns of entry and correction in large vocabulary continuous speech recognition systems. In *Proc. Conf. Human Factors in Computing Systems* (May 1999), pp. 568–575.

[21] KENT DISPLAY: CHOLESTERIC LCD. *http://www.kentdisplays.com*.

[22] L. DENG *et al.* Distributed speech processing in MiPads multimodal user interface. *IEEE Trans. Speech & Audio Processing 10*, 8 (Nov. 2002), 605–619.

[23] LANDAUER, R. Irreversibility and heat generation in the computing process. *IBM J. Research & Development 3* (July 1961), 183–191.

[24] LORCH, J., AND SMITH, A. Using user interface event information in dynamic voltage scaling algorithms. In *Proc. Int. Symp. Modeling, Analysis & Simulation of Computer Telecommunications Systems* (Oct. 2003), pp. 46–55.

[25] MACKENZIE, I. S., AND SOUKOREFF, R. W. Text entry for mobile computing: Models and methods, theory and practice. *Human-Computer Interaction 17* (2002), 147–198.

[26] MICROSOFT SPOT. *http://www.spot-watch.com/*.

[27] MICROSOFT TRANSCRIBER. *http://www.microsoft.com/windowsmobile/downloads/transcriber.mspx/*.

[28] MICROSOFT VOICE COMMAND. *http://www.microsoft.com/windowsmobile/downloads/voicecommand/*.

[29] OMOIGUI, N., HE, L., GUPTA, A., GRUDIN, J., AND SANOCKI, E. Time-compression: Systems concerns, usage, and benefits. In *Proc. Conf. Human Factors in Computing Systems* (1999), pp. 136–143.

[30] PASRICHA, S., MOHAPATRA, S., LUTHRA, M., DUTT, N., AND SUBRAMANIAN, N. Reducing backlight power consumption for streaming video applications on mobile handheld devices. In *Proc. First Workshop Embedded Systems for Real-Time Multimedia* (Oct. 2003).

[31] RAMAN, T. V. *Auditory User Interfaces: Toward the Speaking Computer*. Kluwer Academic Publishers, Boston, MA, 1997.

[32] SEARS, A., AND ZHA, Y. Data entry for mobile devices using soft keyboards: Understanding the effects of keyboard size and user tasks. *Int. J. Human-Computer Interaction 16*, 2 (2003), 163–184.

[33] STARNER, T. Keyboards redux: Fast mobile text entry. *IEEE Pervasive Computing*, 3 (2004), 97–101.

[34] WANT, R., PERING, T., DIANNEELS, G., KUMAR, M., SUNDAR, M., AND LIGHT, J. The personal server: Changing the way we think about ubiquitous computing. In *Proc. Int. Conf. Ubiquitous Computing* (Sept.-Oct. 2002), pp. 194–209.

[35] ZENITHAL: BISTABLE DISPLAYS. *http://www.zbddisplays.com*.

[36] ZHONG, L., AND JHA, N. K. Graphical user interface energy characterization for handheld computers. In *Proc. Int. Conf. Compilers, Architecture, & Synthesis for Embedded Systems* (Nov. 2003), pp. 232–242.

[37] ZHONG, L., AND JHA, N. K. Dynamic power optimization for interactive systems. In *Proc. Int. Conf. VLSI Design* (Jan. 2004), pp. 1041–1047.

Turducken: Hierarchical Power Management for Mobile Devices

Jacob Sorber Nilanjan Banerjee Mark D. Corner Sami Rollins†

Department of Computer Science
University of Massachusetts, Amherst, MA
{sorber, nilanb, mcorner}@cs.umass.edu

†*Department of Computer Science*
Mt. Holyoke College, South Hadley, MA
srollins@mtholyoke.edu

Abstract

Maintaining optimal consistency in a distributed system requires that nodes be always-on to synchronize information. Unfortunately, mobile devices such as laptops do not have adequate battery capacity for constant processing and communication. Even by powering off unnecessary components, such as the screen and disk, current laptops only have a lifetime of a few hours. Although PDAs and sensors are similarly limited in lifetime, a PDA's power requirement is an order-of-magnitude smaller than a laptop's, and a sensor's is an order-of-magnitude smaller than a PDA's. By combining these diverse platforms into a single integrated laptop, we can reduce the power cost of always-on operation. This paper presents the design, implementation, and evaluation of Turducken, a Hierarchical Power Management architecture for mobile systems. We focus on a particular instantiation of HPM, which provides high levels of consistency in a laptop by integrating two additional low power processors. We demonstrate that a Turducken system can provide battery lifetimes of up to *ten times* that of a standard laptop for always-on operation and *three times* for a system that periodically sleeps.

Categories and Subject Descriptors

D.4.7[**Operating Systems**]: Organization and Design—*distributed systems, interactive systems, real-time and embedded systems*; D.4.8[**Operating Systems**]: Performance—*measurements*; D.4.4[**Operating Systems**]: Communications Management—*network communication*;

General Terms

Management, Measurement, Performance

Keywords

Power management, energy management, pervasive computing, mobile computing, low-power computing, embedded devices.

1 Introduction

The performance and utility of any distributed system is impacted by the availability of the participating nodes. In order to execute tasks remotely and maintain consistency of distributed data stores, nodes must be powered on and connected to one another. These requirements are difficult to support in a wired environment; if the participating nodes are mobile, it becomes even more of a challenge. It is particularly difficult to ensure that a mobile node remains *always-on* to participate in the system.

Mobile devices are unique in that they have finite lifetimes. In larger mobile devices, such as laptops, aggressive power management is often used to extend device lifetime by reducing the amount of time the device remains on. Although PDAs and sensors are similarly limited in lifetime, a PDA's power requirement is an order-of-magnitude smaller than a laptop's and a sensor's is another order-of-magnitude smaller than a PDA's. However, these reduced power draws come at the price of reduced functionality and computational power.

This paper presents the design, implementation, and evaluation of Turducken[1], a mobile device architecture that enables full device functionality, always-on availability, and extended device lifetime. Turducken integrates several mobile computing platforms that operate at different power levels into a single multi-tiered device that can operate at the power level of any one of its tiers. While the system supports all of the functionality of its highest power tier, it can utilize lower power tiers to execute simpler tasks, thus reducing the system-wide power consumption and extending the system lifetime. Moreover by integrating an always-on tier such as a sensor we can achieve always-on availability.

Because maintaining consistency of distributed data stores is one of the most integral tasks for mobile distributed systems, we focus our attention on Turducken's

ability to maintain high levels of consistency. Our evaluation compares several Turducken configurations running three common, data-driven applications: time synchronization, web caching, and email. Our results indicate that a Turducken system that integrates an x86-based laptop with a StrongARM and a sensor provides the same level of consistency as a standard laptop computer; however, it can remain always-on for up to *ten times* as long, and *three times* as long if the laptop periodically sleeps to conserve energy. Additionally, we present a theoretical analysis of the lifetime gain of using a Turducken system to execute any task. This analysis demonstrates that Turducken is useful for a broad set of distributed services.

In Section 2 we provide further motivation and introduce the Turducken approach. Section 3 describes the design of the hardware components as well as the software architecture. Section 4 presents a prototype implementation, which we evaluate in Section 5. Section 6 presents related work, and we conclude in Section 7.

2 Motivation

2.1 Consistency in Mobile Systems

A fundamental goal in mobile distributed systems is providing consistency between data stores. Distributed file systems, databases, and applications such as email and the web demand that a user's local view of data be consistent with the view at other nodes in the system. This consistency is ensured through frequent synchronization between nodes. For two end-points to maintain optimal consistency, they must both be *always connected* and *always powered on*. Unfortunately, if either node is mobile, the system cannot make this guarantee and consistency is sacrificed.

The lack of a network connection between two nodes is primarily attributable to physical proximity and wireless network coverage. Network partitions can also be the result of several other factors, including: firewalls; integration of inexpensive short-range wireless connections in consumer devices; or location-based services that intentionally make services only available in specific physical locales. While an end system can attempt to mask these disconnections, it can do little to affect the infrastructure that provides connectivity.

Even if a network path does exist between two endpoints, in a mobile system there is no guarantee that both nodes will be powered on. Mobile nodes have a finite energy supply and thus a finite lifetime. A node may be off because it has exhausted its battery supply, because it has intentionally powered down to conserve energy, or because the user has turned off the device. In any case, if the node performing synchronization or the node with the most recent version of the file is not on then synchronization cannot occur. Traditionally, mobile systems address these problems by attempting to mitigate their effects. For example many systems cache and buffer updates and opportunistically perform synchronization when nodes are powered on and connected. Similarly, many systems support weak-consistency models. This ensures that the system can be used locally even if nodes are disconnected. In essence these techniques allow the system to function even if data stores are not consistent; however, as data stores become increasingly inconsistent, they also become less useful.

2.2 Energy Management Approaches

To achieve high levels of consistency, mobile nodes must be powered on as much as possible so they may take advantage of network connectivity when it exists and may perform synchronization as frequently as possible. This requires that a device be on and consuming energy, even when no useful tasks can be accomplished. For instance, ensuring that a user's mail is immediately delivered to a mobile device requires that the device to be powered on, even when no new mail is arriving. This approach can be very energy inefficient, thus negatively impacting the lifetime of the system.

One approach to reducing energy consumption is to leave the mobile device in an *always-on* mode, but turn off the screen, aggressively spin down the disk [4, 3, 8], scale the CPU voltage and frequency [30, 7, 5, 15, 5], manage wireless interface usage [1], turn off banks of RAM [12, 16, 9, 18], and recompile programs for low power operation [28]. Unfortunately, we observe that a sample laptop using many of these methods only has a lifetime of approximately 8 hours and a standard PDA only lasts for 14 hours. To keep a device in an always-on state requires the user to charge it several times a day, even if it is not actively used. These low-power modes were designed to save power while providing interactivity, not to enable always-on functionality.

Another approach is to *suspend* the device, refreshing only the RAM, and wake up at periodic intervals to perform synchronization (e.g., download web updates). For instance, to extend the lifetime of an IBM Thinkpad to 3 days, we can wake the laptop for approximately 2 minutes of every hour. However, there is a trade-off between the frequency with which we wake the device and the level of consistency maintained: waking up more often costs more energy, but provides higher consistency. Additionally, there is no guarantee that a device will be in range of a network and able to perform synchronization when it wakes. An approach such as Wake-on-Wireless [24] can reduce the amount of energy spent waking a device if no network connectivity exists. How-

ever, a significant amount of energy is still wasted if a high-power device, such as a laptop, wakes to discover that a network connection exists but no updates are ready (e.g., no new mail has arrived or a cached web page has not changed).

While these approaches provide considerable energy savings, they are inappropriate for extending the maximum lifetime of the device while providing high consistency. This is because they fail to address the *non-reducible* power of mobile devices [14], which dominates the lifetime of the battery. The reducible power is the amount of power that can be eliminated from a running system while maintaining the ability to do computation. Common sources of non-reducible power include the power supply, the on-board oscillators, the memory and I/O buses, and the limited range of frequency and voltage scaling [2]. Some small embedded systems have proposed using multiple processor cores that can be shut off [19, 11]. However, such a system only reduces the power draw of the processor, which constitutes less than 10% of the power consumed by a laptop [2]. Even the most highly-optimized laptop computer incurs a significant energy cost to wake up and download a piece of data.

2.3 A New Approach: Turducken

The amount of non-reducible power varies for different devices. For example, the non-reducible power of a StrongARM-based PDA is on the order of twenty-times smaller than the non-reducible power of an x86-based laptop. As another example, the non-reducible power of a small sensor is significantly smaller than that of a device such as a wireless music player. Typically, devices are carefully optimized to provide their promised functionality at the lowest possible energy cost, and devices that provide less functionality have smaller non-reducible power. Fortunately, there is significant overlap in the functionality provided by high-power and low-power devices. For example, maintaining a consistent view of a file requires only the ability to connect to a network and download data; a variety of devices can provide this functionality.

The goal of our approach, Turducken, is to reduce the energy cost of maintaining high levels of consistency on mobile devices by combining several optimized mobile platforms into one integrated system. By combining a very low-power platform such as an ATmega-based sensor with a very high-power platform such as a laptop, we can produce a system that can be always-on and still have all of the functionality of a laptop computer.

The system is composed of a set of *tiers*, each with a set of capabilities and a power mode. The system as a whole executes tasks (e.g., downloads data updates) by waking the tier that has the capabilities to execute the task in the most efficient manner. For example, one tier might include a StrongARM processor, along with its memory and storage. This tier could be integrated with a standard x86-based laptop. We can then suspend the x86-tier and rely upon the StrongARM-tier to wakeup and perform periodic tasks.

For instance, if the StrongARM-tier wakes up periodically to cache a copy of frequently-used web pages, when the user opens the laptop, those pages will be available and highly consistent. If the laptop alone were to frequently wake itself up and cache those same pages, it would attain the same level of consistency; however, the overall *lifetime* of the system would be greatly diminished.

Note that in this integrated system all of the tiers use a common battery, are connected by a common bus, and effectively form a tightly coupled distributed system. However, from the user's perspective it appears to be a normal laptop. The addition of extra components does increase the weight and cost of a mobile system. For instance, adding a StrongARM mobile processor and memory to the inside of a laptop may add $100 and a few ounces. However, the extra capabilities the system provides outweigh these costs. Another observation is that this system could be commercially built using commodity components. The architecture is fully composable: any set of tiers can be used together to give a wide variety of power modes and can be applied to many mobile devices.

2.4 Lifetime Gains

We can demonstrate Turducken's effectiveness using a simplified analysis of the expected gains in the lifetime of the system. Here we analyze the expected lifetimes of two different systems. The first is a normal laptop that wakes up periodically to synchronize and goes back to sleep. The second is a simplified Turducken system that consists of an x86-tier integrated with a StrongARM-tier; the x86-tier remains suspended, while the StrongARM-tier periodically wakes up and performs the same synchronization task.

Our analysis shows that there are two circumstances in which Turducken provides gains in the system lifetime: 1) if the fraction of time the laptop spends awake is large enough to overcome the extra burden of the StrongARM-tier's suspension power, and 2) if the time in which the StrongARM-tier can perform the synchronization task is a reasonable multiple of the time the x86-tier takes.

The first equation details the lifetime of a laptop that wakes at periodic intervals:

$$L_L = \frac{C}{f_A^L \cdot P_A^L + (1 - f_A^L) \cdot P_S^L}, \quad (1)$$

where C is the capacity of the battery, f_A^L is the fraction of time the laptop spends awake, P_A^L is the power it expends while awake, $1 - f_A^L$ is the fraction of time the laptop spends asleep, and P_S^L is the power it expends while suspended.

The lifetime of a Turducken system, consisting of an x86-tier paired with a StrongARM-tier, can be represented as:

$$L_T = \frac{C}{f_A^P \cdot P_A^P + (1 - f_A^P) \cdot P_S^P + P_S^L}, \quad (2)$$

where f_A^P and P_A^P are the fraction of time and power the StrongARM tier spends awake, and $1 - f_A^P$ and P_S^P are the fraction of time and power the StrongARM tier spends suspended respectively. C is the battery capacity. The x86-tier remains suspended while the StrongARM-tier wakes up. Thus the x86-tier expends P_S^L all of time.

Using these two equations we can express the gain of the Turducken system as the ratio of the lifetime of the Turducken system to that of a standard laptop:

$$\frac{L_T}{L_L} = \frac{f_A^L \cdot P_A^L + (1 - f_A^L) \cdot P_S^L}{f_A^P \cdot P_A^P + (1 - f_A^P) \cdot P_S^P + P_S^L}. \quad (3)$$

As long as this ratio is greater than one, Turducken has a positive impact on the lifetime of the system. Using the proof found in the appendix and a set of measurements taken from a prototype system running the web caching application, described in Section 4, we find that if the web caching application only runs 17 seconds of every hour, then the StrongARM-tier can perform the synchronization task up to 5 times slower than the x86. In fact, because web caching is network bound, the ratio of execution time is actually one-to-one, and for reasonable levels of consistency the web caching application runs much more than 17 seconds of every hour. Because of these two factors, Turducken typically provides an increase in lifetime substantially greater than this lower bound.

The remainder of this paper describes the design and prototype implementation of the hardware and software components of our system. Additionally, we present a set of experiments which quantify the benefits of using the Turducken system.

3 System Design

The design of a Turducken system is composed of three parts: the hardware, the underlying system architecture, and the model for distributing applications across the tiers. In general, the design is similar to many distributed systems; each tier is under autonomous control while decisions are made in a distributed manner. Client applications reside at the most powerful tier, and tasks that support those applications are distributed among the various tiers.

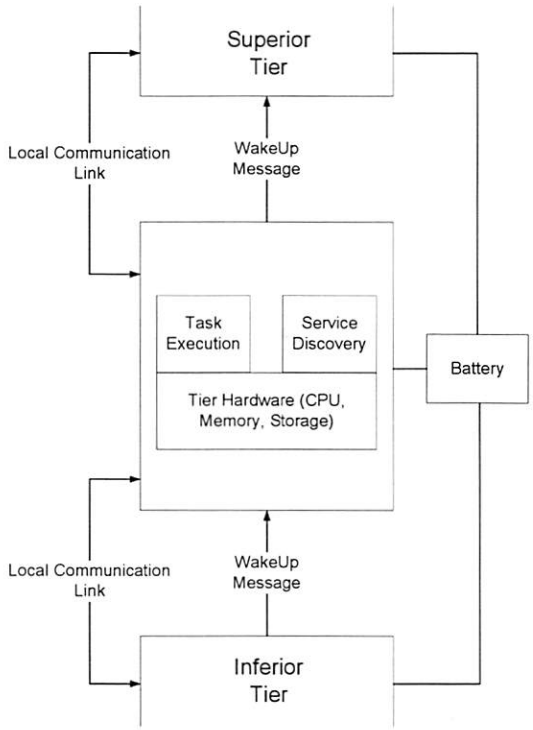


Figure 1: System Design

3.1 Hardware Design

A Turducken system is designed in a strictly hierarchical manner, and each tier is more powerful than any tier below it. Each tier can communicate with a *superior* tier and an *inferior* tier—the two exceptions being the top and bottom of the hierarchy. Communication occurs via a local communication network and the tiers are connected to a common power source. Moreover, each tier has the ability to draw its superior tier out of a suspended mode. It is fully composable; the system will still operate correctly if tiers are added, removed, or changed. This provides a flexible architecture that can accommodate the evolving number of hardware platforms available in low-power computing. An overview diagram of our design is shown in Figure 1.

Each tier contains an independent processor, memory, internal bus, and persistent storage system. Each may also have an independent external wireless network interface, although these can be shared by routing through the

inter-tier communication network. The set of tiers can be architecturally homogeneous and span a range of power requirements. By limiting the interface between tiers, we achieve composability. Integrating new tiers with differing instruction sets, capabilities, operating systems, and power requirements is straightforward.

A Turducken system is also fully autonomous and does not depend on any special hardware from the external network. For instance, Turducken does not require external networks to be equipped with hardware wakeup signals, such as those used in the Wake-On-Wireless project [24]. This ensures that the system will work with high-powered access points, as well as low-power, peer-to-peer, wireless devices. Because there is no dependence on the external, wireless networking hardware, Turducken will work with any radio interface, as well as accommodate multiple radios in the same system.

3.2 System Architecture

The system as a whole is responsible for accepting tasks from the user and executing them in a way that extends the lifetime of the system. Tasks can be anything from keeping the time synchronized to ensuring that the local copy of the user's email is current. The user, or a service executing on behalf of the user, introduces tasks at the highest level and the system distributes these tasks among the different tiers in a way that extends the lifetime of the overall system. Each tier is capable of several operations: perform tasks or discover services; inform other tiers when necessary; and manage its local consumption of power. We discuss each responsibility in more detail below.

Perform a task. A tier can perform a task if the required service is reachable and ready to be used. Ideally, a task will be executed by the most efficient tier capable of performing that task. For example, the highest-power tier would be required to synchronize a very large media file while a StrongARM-tier can perform the task of synchronizing a cache of web pages. For some applications, a tier will also need to pass the results of task execution to its superior tier. For example, a web page cached by a StrongARM-tier will ultimately be delivered to the highest tier in response to a user request.

Perform service discovery. A tier can also monitor the availability of a service required by a higher-power tier in order to perform a task. Service discovery may simply discover the existence of a service, or it may determine whether a particular service needs to be used (e.g., whether or not a user has new email that needs to be fetched). Again, service discovery should be performed by the lowest-power tier that is capable of discovering the service. In many cases, it is also possible to further decompose service discovery. For example, to determine if a large media file is available to be synchronized, an ATmega-tier can monitor the network for connectivity, a StrongARM-tier can determine if the file has changed, and the x86-tier can actually perform the task.

Enter a suspension state. If a tier is not needed to perform a task or service discovery, it may put itself to sleep in order to conserve energy. In some cases, this may require that the tier delegate tasks or service discovery jobs to its inferior tier. For example, a StrongARM-tier may notify an ATmega-tier that it is going to sleep and needs to be woken when a network connection is available.

Wake its superior tier. Once a tier has discovered an appropriate service, it may need to wake its superior tier so that it can perform the task. Each tier is capable of waking its superior tier. In this way, a tier can rely on its inferior tier to tell it when there is something to do rather than requiring the system to wake periodically and check.

3.3 Distributing Applications

There are several methods of distributing application responsibilities over the tiers. We describe each of these options here:

System-Aware Architecture. The first option is to build an application that is customized for the system. Such an application requires designers to create application components for each tier. In addition, the application must define the messaging protocol used to communicate between components. This hand-coded option is useful for new applications and also for applications, such as time synchronization, which are fairly simple to implement.

Proxy-Based Architecture. A second option is to use a proxy-based architecture that can take advantage of existing distributed application components. Using this architecture, a tier that executes tasks appears as a proxy service provider or a replicated server to superior tiers. Many distributed applications, such as distributed file systems, email, and web caching, already support this design. Therefore, the advantage is simplicity and deployability—proxies only require recompiling and reconfiguring the application rather than recoding. Unfortunately, not all applications will tolerate a proxy that queues responses, requiring some modification of applications. One

possibility is to use queued RPC as found in the Rover toolkit [10].

Transparent Architecture. A final option is to develop a Turducken system component that is capable of transparently migrating application processes. One way to support this is by using traditional process migration [13, 21, 26, 27]. We have eliminated this as a possibility due to the complications arising from different architectures, operating systems, and memory capacities. Another possibility is to use virtual machines, either programming-language virtual machines such as those used for Java, or a lightweight, OS-level virtual machine such as Denali [31]. Unfortunately, the current lack of a virtual machine that runs on all of these platforms, and the vastly differing capabilities of the tiers makes this difficult.

4 Prototype Implementation

To demonstrate the efficacy of our approach, and to provide a test platform for our work, we have built a prototype Turducken system. The prototype currently consists of a hardware implementation and three applications: time synchronization, web caching, and IMAP synchronization.

4.1 Hardware Implementation

The hardware prototype, shown in Figure 2, consists of three tiers: an x86-based IBM Thinkpad X31, a Compaq iPAQ 3870 StrongARM-based PDA, and a CrossBow Mica2Dot ATMega-based Mote. The Mote and iPAQ are directly connected via a serial interface and the iPAQ and the laptop are directly connected via a USB interface. The Mote can wake the iPAQ through the use of the serial DCD line, and the iPAQ can wake the laptop by sending a request to the Mote, which wakes the laptop by triggering a relay connected to the keyboard. Our prototype can currently be reconfigured as: x86, x86+ATMega, or x86+StrongARM+ATMega. Each tier also contains a real-time clock (RTC) that can generate a wake interrupt. If we reconfigure the system as x86 only, it can suspend itself and use its RTC to wake it at set intervals.

This prototype differs from our design in three significant ways. First, the hardware components are all physically separate—a deployed system would integrate all of the components into a laptop form-factor. The connections shown in the picture would all be internal to the system. Second, there is a plethora of extra parts in our prototype. An integrated implementation would eliminate much of the PDA, including its screen, sleeve, and buttons. Third, each tier is run from its own battery. The Turducken design assumes that there is only a single, shared battery. This has implications for how we evaluate the system.

	Execution Tier	Incoming or Outgoing
Time Sync	$\geq$ATmega	Incoming
Web Cache	$\geq$StrongARM	Incoming
IMAP Sync	$\geq$StrongARM	Both

Table 1: This table shows a summary of the application characteristics. The execution tier denotes where the application is carried out, and Incoming or Outgoing describes the direction of updates.

In our implementation, there are two types of wireless interfaces: WiFi and the Mote's custom radio interface. There are both advantages and disadvantages to having access to multiple wireless standards. It does allow the system to take advantage of a broader range of services by allowing it to communicate with more devices; however, it makes system design more challenging since certain tasks may require a particular network interface and thus it cannot be accomplished by all tiers. To mitigate this disadvantage, we have attached a WiFi detector to the Mote. The detector can determine if WiFi signals are present, though it cannot communicate using WiFi or discover if an access point is open or closed.

Even though the x86 and StrongARM tiers each have WiFi interfaces, there is no reason to use them both in the Turducken system. In a configuration that includes both, we turn off the x86-tier's interface and route all traffic through the StrongARM-tier. This saves power, thus extending the battery lifetime of the system.

4.2 Applications

We have developed and deployed three applications that are representative of commonly-used mobile distributed services: time synchronization, web caching, and IMAP synchronization. Time synchronization is necessary for timestamping distributed updates and determining timeouts in soft-state protocols. Web caching on mobile devices allows the mobile node to serve pages during periods of disconnection and improves response time when connected. IMAP synchronization maintains a local mail cache that can serve mail during periods of disconnection and improves response time. In addition, a local IMAP store can buffer outgoing mail and send it when the node is connected.

These applications also represent three broader classes of applications. These classes are defined by the traits listed in Table 1. Time synchronization represents appli-

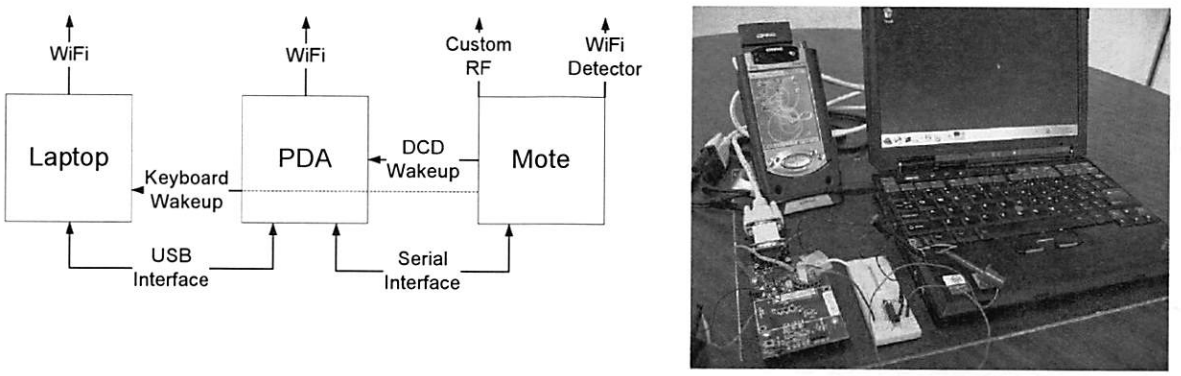


Figure 2: These figures show the prototype implementation of the Turducken System. The diagram on the left shows the logical connections between components and the photo on the right shows the current prototype.

cations that require limited processing and limited transmission of incoming data updates. Web caching represents applications that require more significant processing and larger amounts of incoming data. This is similar to a variety of publish-subscribe systems. IMAP synchronization represents applications that require fairly significant processing and support for outgoing as well as incoming updates. This is similar to the requirements of distributed file and database systems, though the consistency requirements are not as strict.

4.2.1 Time Synchronization

The time synchronization application follows the system-aware programming model. The ATmega-tier runs a custom built Network Time Protocol (NTP) client that synchronizes its local clock with a known time server every t seconds. The StrongARM and x86 tiers can then request the current time from the ATmega-tier and update their local clocks. We define an explict API for this communication. When the ATmega-tier is not present, the the x86-tier uses its RTC to wake every t seconds and synchronize with the remote time server using the UNIX utility `ntpdate`.

4.2.2 Web Cache

The web cache application follows a proxy-based programming model. The ATmega-tier detects the presence of a WiFi signal; the StrongARM-tier runs a Squid proxy cache; and the x86-tier runs a web browser. Every t seconds, the ATMega determines whether a WiFi connection is available and, if so, wakes the StrongARM-tier. The StrongARM-tier remains awake for 30 seconds while the proxy continuously fetches expired cache items. Web requests originating from the web browser running on the x86-tier are routed through the StrongARM-tier. These requests can be transparently serviced by the proxy when no network connection is available.

When the StrongARM-tier is not present, the Squid proxy runs on the x86-tier and the cache is stored on the system's hard disk. The ATmega-tier or RTC wakes the x86-tier every t seconds. If a connection is present, it remains awake for 30 seconds while the Squid proxy fetches expired cache items. Again, the Squid proxy can transparently fulfill requests from a web browser.

4.2.3 IMAP Synchronization

The IMAP synchronization application also follows a proxy-based programming model. The ATmega-tier detects the presence of a WiFi signal and the StrongARM-tier runs a UNIX utility named `mailsync`, which performs synchronization between an IMAP server and a *secondary* mail store. The x86-tier maintains the *primary* mail store and uses `mailsync` to synchronize with the StrongARM-tier's secondary mail store. The x86-tier also runs the user's mail client. Every t seconds, the ATmega-tier determines whether a WiFi connection is available and, if so, wakes the StrongARM. The StrongARM-tier uses `mailsync` to retrieve incoming mail from and send outgoing updates to the user's mail server. Incoming mail is stored in the secondary mail store hosted on the StrongARM-tier.

When the user turns on the x86, it synchronizes its primary store with the secondary store on the StrongARM-tier. The user accesses mail by configuring the mail client to point to the primary mail store on the x86-tier. When the user suspends the x86-tier, any changes the user has made will be synchronized with the StrongARM-tier which will synchronize with the remote mail server when connected.

In some cases, the user may receive pieces of mail that are too large to be stored in the StrongARM-tier's flash memory. To accomodate this scenario, the primary

mail store also synchronizes with the remote mail server when possible. In addition, we would like to modify the StrongARM-tier to wake the x86-tier when it detects this situation, though we have not yet implemented this feature.

If the StrongARM-tier is not present, the x86-tier synchronizes directly with the remote mail server when connected. Similar to the web cache, the ATmega-tier or RTC wakes the x86 every t seconds. If the x86-tier discovers that no connection is present, it goes back into a suspended mode without performing synchronization.

Both the IMAP synchronization and the web caching applications were implemented using standard components. Due to the distributed nature of these applications, recoding is not necessary in order to deploy them on our prototype Turducken system. Each component can simply be recompiled for both the x86 and StrongARM architectures.

5 Evaluation

The primary goal of Turducken is to extend the lifetime of a mobile computing device while allowing it to remain aware of its environment when not actively in use. In our evaluation of the Turducken system, we measure the lifetime of several Turducken configurations running the following three sample applications: time synchronization, web caching, and IMAP synchronization. For each application, we compare the system lifetimes of different configurations with respect to data consistency. Finally, we focus on the web caching application and compare system performance with respect to variable network and service availability.

5.1 Methodology

Our evaluation measures the lifetime of several system configurations running varied workloads. Measuring the lifetime of a Turducken system presents a number of interesting challenges. Explicitly measuring the lifetime of a single configuration running a single workload can take longer than a week. Collecting even a small number of data points using this method is impractical with only a single prototype system. We address this problem using time dilation. For each experiment we measure the energy consumed by the system under a small number of workloads. Using these measured values, we use extrapolation to project system lifetimes over a wide range of data points.

We calculate the lifetime of the system from the average energy that is required to run a particular application under a given workload for a set period of time. From this value we can estimate how long it will take the system to drain a battery of known capacity. We also make the assumption that the power draw of a full system will be no greater than the sum of the power draw of each tier. This estimate is conservative since an integrated system can use more power-efficient communication links between tiers.

For the experiments presented here, we measure the amount of energy consumed by each tier using the tier's native power management interface. Batteries used in modern mobile devices typically contain a gas gauge chip, such as the Texas Instruments BQ2011 chip used in the x86-tier's battery, which considers temperature, battery chemistry, and past usage to accurately compute the amount of energy remaining in the battery. This approach is sufficient to measure the average energy consumtion over a particular period of time. This is similar to the method used to measure power consumption of the Odyssey System [6].

Using this method we measure the energy consumed by each tier over a fixed period of time, and calculate the amount of time it takes the entire system to drain a full battery. This calculation depends on several factors: the average power draw of each tier while active; the average power draw of each tier while suspended; the amount of time each tier spends active; and the amount of time each tier is suspended.

We measure the power draw of both the x86-tier and the StrongARM-tier in suspended mode over a 10 hour period of time. The energy in the battery is sampled immediately before and after the period of suspension in order to determine the total energy consumed. We divide this value by the total experiment time to obtain the power draw of each device in suspended mode. For the StrongARM-tier, we obtain the full battery capacity from the manufacturer's specification. For the x86-tier we use the estimated capacity specified by the device's battery.

To determine the power draw of the x86-tier and StrongARM-tier in active mode we run each application on each system configuration for a 24-hour period. During all experiments, we turn off both the screen and backlight of the two higher tiers in order to make a more fair comparison. For each device, we measure the amount of time it is active, t_A, the amount of time it is suspended, t_S, and the total energy, E, consumed by the tier. Using the total amount of time suspended and the suspended power draw, P_S, we calculate the energy consumed while suspended, E_S over the 24 hour period:

$$E_S = P_S t_S. \tag{4}$$

We then use the total energy, E, and the energy used while suspended, E_S, to compute the energy used while active:

$$E_A = E - E_S. \tag{5}$$

Mode	x86	ATmega
Active (mW)	11,600	26.4
Suspended (mW)	180	0.056

Table 2: The active and suspended power consumption of each tier running the time application. The active power consumption for the StrongARM-tier was not measured since it never synchronizes with the time server.

Mode	x86	StrongArm	ATmega
Active (mW)	10,955	740	26.4
Suspended (mW)	180	40	0.056

Table 3: The active and suspended power consumption of each tier running the web caching application.

By dividing the energy used while active by the amount of time the system is active, we obtain the power draw, P_A, of each tier in the active state:

$$P_A = \frac{E_A}{t_A}. \quad (6)$$

The resulting power draws are shown in Tables 2, 3, and 4.

For the ATmega-tier, we assume it will be always on and establish a generous upper bound on the power draw from the Crossbow datasheets. Even using this upper bound, the power draw of the ATmega-tier has very little impact on the lifetime of the system.

Using these individual measurements, we calculate the power draw of the full system as the sum of the power draw of each tier in the appropriate state. Using this value, we calculate the amount of time it takes the entire system to drain the entire battery of the x86-tier.

5.2 Consistency

The goal of our first set of experiments is to vary the level of consistency required and observe the consequent lifetimes of several system configurations. To accomplish this, we vary the interval at which the system wakes to perform synchronization from 0 (always on) to 0.5 hours. A wake interval of i minutes ensures that data is inconsistent for no longer than i minutes.

For each of these experiments, a wireless network is always present, the remote service is available, and new data updates are ready. For the time synchronization application, we assume that the time is synchronized whenever the system wakes. For the web caching application, the system maintains a 5 MB cache consisting of 15 web sites. For the IMAP synchronization application, the Turducken system fetches data updates and sends any

Mode	x86	StrongArm	ATmega
Active (mW)	11,720	810	26.4
Suspended (mW)	180	40	0.056

Table 4: The active and suspended power consumption of each tier running the IMAP synchronization application.

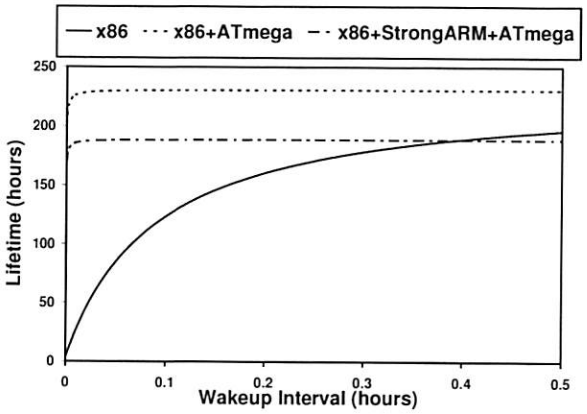


Figure 3: The lifetime of three system configurations running the time synchronization application. As the system wakes more frequently, Turducken provides a more significant gain in lifetime.

queued, local updates upon waking. For this experiment, the x86-tier wakes for 2 minutes of every hour to simulate a user creating modifications to the local mail store. This store initially contains 4MB of mail in four separate folders. The queued updates to the local store are sent to the remote IMAP server when the StrongARM-tier wakes to synchronize. In addition, new mail is sent to the inbox at a rate of 120KB per hour. During synchronization, the Turducken client fetches this mail.

The results of the time synchronization experiment are shown in Figure 3. When the system synchronizes frequently, the lifetime of the x86-only system degrades drastically while both the x86+StrongARM+ATmega and x86+ATmega configurations maintain nearly constant lifetimes. This is a consequence of the fact that when using a Turducken system, the x86 and StrongARM tiers never need to come out of a suspended state. In this case, the x86+ATmega configuration has a lifetime of about 225 hours and the x86+StrongARM+ATmega has a lifetime of approximately 180 hours. The difference between these two configurations is a result of the energy draw of the StrongARM-tier in suspended mode.

Figure 4 shows the results of the web caching experiment. We observe that the x86+StrongARM+ATmega

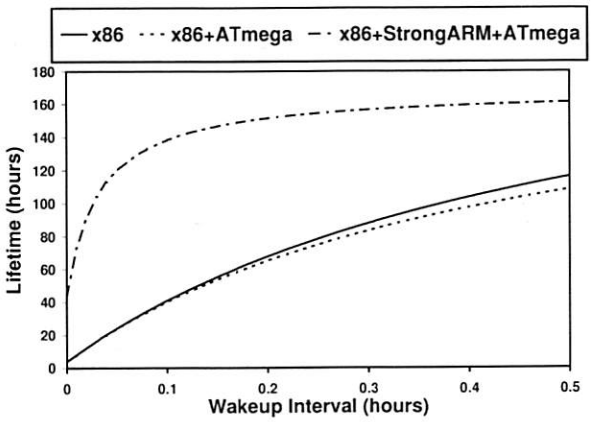


Figure 4: The lifetime of three system configurations running the web caching application. For this application, the full three-tiered Turducken system offers up to a 4 times longer lifetime and consistently performs better than the x86-only configuration.

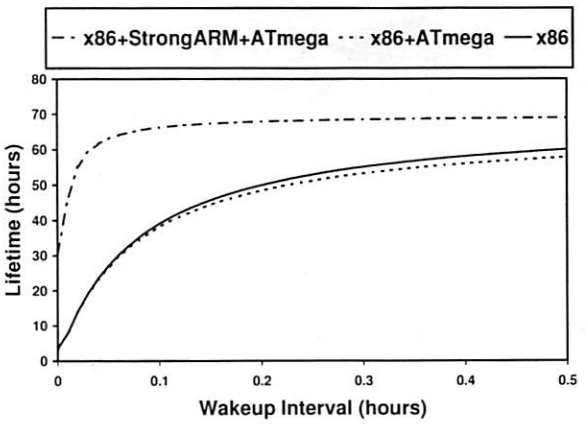


Figure 5: The lifetime of three system configurations running the IMAP synchronization application. For this application, the full Turducken system offers a 1.5 times longer lifetime and consistently performs better than the x86 only configuration.

consistently performs better than the other configurations, providing a ten times improvement for always on operation and a three times improvement when the system wakes up every six minutes. Additionally, we observe that as the wake interval grows, the lifetime gain lessens. This is a result of the energy required to power the StrongARM-tier in suspended mode. Similarly, the x86+ATmega system performs worse than the x86-only configuration for larger wake intervals because of the additional energy required to power the ATmega tier. Again, we can conclude from these observations that the higher the level of consistency required, the better the performance of Turducken.

Figure 5 shows the results of the IMAP synchronization experiment. The relative performance for IMAP synchronization is very similar to the web caching application, however, we observe that the absolute system lifetimes are significantly smaller. This is a result of the workload of IMAP synchronization. This particular experiment requires that the x86-tier wake periodically to simulate a user updating the local mail store, which costs additional energy. This application also introduces additional outgoing network traffic which impacts energy usage. However, we still observe that Turducken enjoys at least a 150% improvement in system lifetime for wakeup intervals less than six minutes. If the x86-tier does not perform periodic synchronization and only wakes up once an hour to send and receive updates its average lifetime is found to be 75 hours. However, the cost of this gain in system lifetime is that the expected time to get an update is $\frac{1}{p}$ hours, where p is the probability of a network connection being available. Since this latency can be large for small values of p, it is reasonable to sacrifice 13% of the system's lifetime in exchange for one-tenth the expected latency.

Figure 6 shows the average power draw for each tier. Each bar represents the total average power consumed by a particular configuration running a particular application. A bar is composed of several components that show each tier's contribution to the average power draw of the entire system. We further decompose each tier's contribution into its active and suspended modes. For example, for the x86-only configuration running the time application, the graph shows that the x86-tier spends most of its time suspended and a small amount of time in its active mode. Similarly, when it is augmented with an ATmega-tier, it spends all of its time suspended and the ATmega-tier expends a negligible amount of power. In the web caching experiment, the x86+StrongARM+ATmega configuration is able to replace the active power of the x86-tier with the StrongARM-tier. The mail experiment sees a similar gain; however, because the x86-tier spends more time in active mode, the resulting active power draw is larger. We observe that Turducken systems achieve lower average power consumption by replacing active power consumption in less efficient tiers with more efficient ones.

5.3 Network and Service Availability

The goal of our second set of experiments is to vary the availability of a wireless network and the availability of the required service, and observe the consequent lifetimes of several system configurations. For this set of

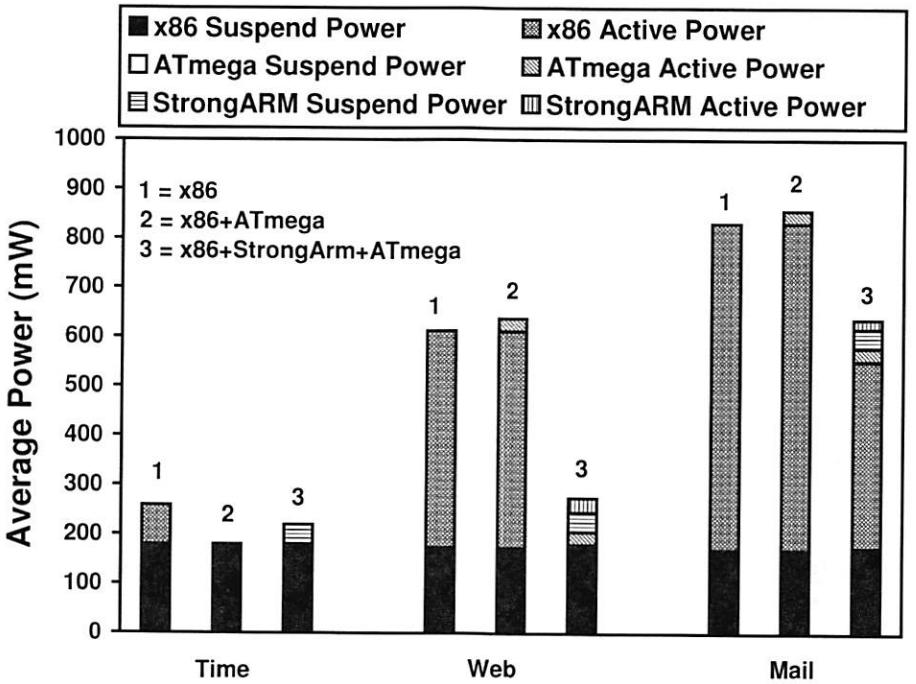


Figure 6: This figure shows how each tier, in different states, contributes to the average power draw of the system as a whole. We observe that Turducken systems achieve battery lifetime gains by replacing active power consumption in less efficient tiers with more efficient ones.

experiments, we look exclusively at the web caching application and fix the wake interval at 12 minutes. In the first experiment, we vary the probability that a wireless network is available from 0 (network never available) to 1 (network always available). In the second experiment, we fix the probability of wireless network availability at 1 and vary the probability that a set of web servers is reachable from 0 (web servers never reachable) to 1 (web servers always reachable). For this experiment, we assume that either all web servers are reachable or no web servers are reachable and we assume that it takes a trivial amount of time to determine reachability for all servers.

The results of varying the network availability are shown in Figure 7. When the probability of WiFi is low, the x86+ATmega system performs best. This is because it can avoid waking the x86-tier if no signal is present. The x86+StrongARM+ATmega system enjoys the same benefit, but incurs the cost of powering the StrongARM-tier in suspended mode. Interestingly, the x86-only configuration performs similar to the x86+StrongARM+ATmega for low probabilities. This implies that the cost to periodically wake the x86 to discover that no network is present is roughly equivalent to the cost of powering the StrongARM and ATmega tiers in suspended mode. As the probability of a network connection increases, the x86+StrongARM+ATmega system remains nearly constant, outperforming the other configurations by up to a factor of 2. This is a result of the energy saved fetching web pages using the StrongARM-tier without waking the x86-tier. We can conclude that Turducken provides a greater benefit as network coverage increases, and performs no worse than an x86 alone as coverage decreases.

The results of varying the availability of web servers is shown in Figure 8. The results for this experiment are similar to the previous experiment with the exception of the x86+ATmega configuration. While the ATmega-tier can determine the presence of WiFi, it cannot determine the reachability of a web server. Therefore, the ATmega-tier must always wake the x86-tier to determine if the web servers are reachable. This costs the x86+ATmega configuration up to 40 hours of lifetime. However, as the probability of service increases, the benefit of Turducken increases.

5.4 Observations

Our primary observation is simple: for many common distributed applications, a Turducken system can maintain a high level of consistency at a fraction of the power

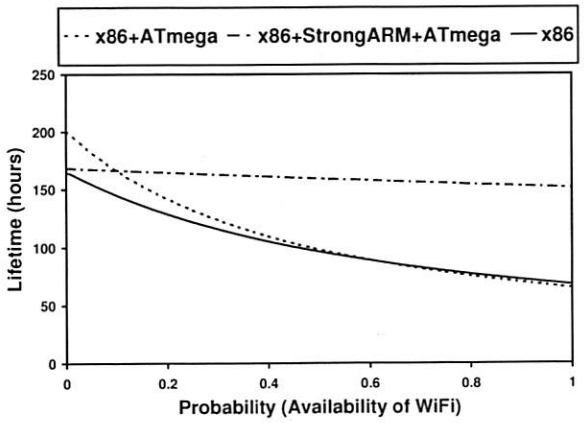


Figure 7: This figure shows the battery lifetime of different configurations with respect to varying the probability of availability of WiFi. As network coverage increases, Turducken provides a greater benefit.

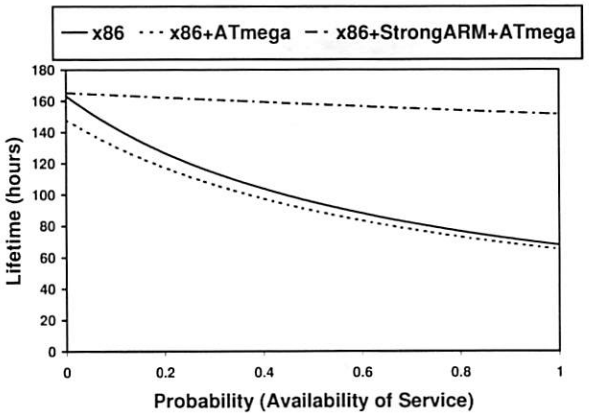


Figure 8: This figure shows the battery lifetime of different configurations with respect to varying probability that a set of web servers is reachable. The benefit of Turducken is evident as the probability that the servers are available increases.

cost of a conventional laptop. This allows system behavior which has traditionally been ruled out in favor of conserving battery power. Naturally, there is a cost incurred when powering additional devices. This cost becomes noticeable when the system wakes up less frequently, reducing the benefit and retaining the cost of the additional hardware. Fortunately, even if the system never wakes up, the x86+StrongARM+ATmega configuration will last 82% as long as the x86-only system.

Our experiments have also shown that the main limiting factor of the system's battery lifetime is the suspended power draw of the x86-tier. Our proposed solution to this is to use hibernation, which involves saving the machine's state to disk and powering it down. When the system is restored, it boots to the previously saved state. Clearly, it will cost more in both energy and latency to wake a device out of hibernation; however, during times of little or no activity (e.g. at night), using hibernation could result in significant power savings, potentially extending the system's lifetime to *over a month* on a single charge.

Additionally, it is clear that the benefit achieved is highly application dependent. For example, in the case of very simple applications, like time synchronization, the x86+ATmega configuration achieves the best performance. The best set of tiers for a particular Turducken system depends on the target applications that the system will host.

6 Related Work

A number of related research projects have explored strategies for reducing energy consumption of mobile devices. The Wake-on-Wireless project (WoW) [24] proposes augmenting a PDA with a wireless sensor. An in-network server notifies the sensor when it should wake the PDA such that it can serve incoming requests. The goal of WoW is similar to the goal of Turducken; low-power operating modes in mobile devices. However, this paper has shown the value in augmenting laptops with multiple tiers that can *execute* synchronization jobs: tiers may perform many operations without waking up the next tier. Also, Turducken is a completely standalone system, not requiring any support from the wireless network. Some work has also looked at integrating multiple radios into a mobile platform [22, 20], and we use this idea in Hierarchical Power Management; however, we are focused on integrating entire independent subsystems rather than individual hardware components. Mayo and Ranganathan proposed energy scale-down as a technique for saving power in mobile devices [17]. They make a similar observation that different mobile devices are optimized for different power points. They specifically suggest using wireless LAN energy management and multiple processor cores, as well as possibly using multiple displays in a mobile device.

Several projects have looked at managing energy from a whole-system standpoint. The Odyssey System [6] trades off resources, such as energy, for application fidelity. The ECOsystem [32] manages energy as any other operating system resource, enforcing fairness be-

tween applications, as well as setting global energy constraints. Simunic, et al. [25] propose a general method to manage energy consumption in across several system components. These systems are primarily designed for making short-term decisions and do not directly address non-reducible power in mobile devices.

An alternative to reducing the energy consumed while utilizing remote services is to ensure that the services are available locally, on the user's personal devices. A number of research projects have focused on ensuring availability of a user's personal data. The Personal Server [29] is a compact storage device which can provide reliable access to a user's personal data. Because the device does not have any kind of display, it operates at a low power point. However, unlike Turducken, the Personal Server provides a specific set of services and does not provide the same level of composability or flexibility in managing energy usage. Another approach is to ensure personal data availability by monitoring devices in a Personal Area Network (PAN), and migrating data from a device when its energy supply becomes critically low [23]. Again, this does not ensure that a device can use services provided outside of the PAN. Additionally, the focus of Turducken is to increase availability for a single, integrated system. However, we expect that the techniques developed for Turducken could also be useful managing energy and availability in a disconnected mobile distributed system.

7 Conclusions

In this paper, we have presented the design and prototype implementation of Turducken, an approach that integrates, into a single system, a series of components that operate at various power levels. Turducken provides both a hardware and software infrastructure that can intelligently use available energy while maximizing device utility. We have demonstrated a prototype implementation and evaluated of its performance. We found that by using additional low-power tiers for synchronization tasks, we can enable greater levels of consistency in distributed services. These techniques give the Turducken system a lifetime that exceeds that of a standard laptop by as much as ten times for always-on operation and three times for less stringent consistency requirements. Until there is a significant improvement in battery technology, strategies like Turducken are imperative for intelligently managing energy.

Acknowledgments

The authors wish to thank Allison Clayton, Brian Levine, and Prashant Shenoy who provided helpful feedback on earlier drafts. This work is supported in part by the National Science Foundation under grants CNS-0447877, DUE-0416863, and EIA-0080119. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the the National Science Foundation or the U.S. Government.

References

[1] ANAND, M., NIGHTINGALE, E. B., AND FLINN, J. Self-tuning wireless network power management. In *Proceedings of the 9th ACM International Conference on Mobile Computing and Networking (MobiCom'03)* (San Diego, CA, September 2003).

[2] CHINN, G., DESAI, S., DISTEFANO, E., RAVICHANDRAN, K., AND THAKKAR, S. Mobile PC platforms enabled with Intel Centrino mobile technology. *Intel Technology Journal 7*, 2 (May 2003).

[3] DOUGLIS, F., KRISHNAN, P., AND BERSHAD, B. N. Adaptive disk spin-down policies for mobile computers. In *Proceedings of the 2nd Symposium on Mobile and Location-Independent Computing* (April 1995).

[4] DOUGLIS, F., KRISHNAN, P., AND MARSH, B. Thwarting the power-hungry disk. In *Proceedings of The USENIX Winter 1994 Technical Conference* (San Francisco, CA, 1994).

[5] FLAUTNER, K., REINHARDT, S., AND MUDGE, T. Automatic performance-setting for dynamic voltage scaling. In *Proceedings of the Seventh ACM International Conference on Mobile Computing and Networking (MobiCom'01)* (Rome, Italy, July 2001).

[6] FLINN, J., AND SATYANARAYANAN, M. Managing battery lifetime with energy-aware adaptation. *ACM Transactions on Computer Systems (TOCS) 22*, 2 (May 2004).

[7] GOVIL, K., CHAN, E., AND WASSERMAN, H. Comparing algorithms for dynamic speed-setting of a low-power CPU. In *Proceedings of the First ACM International Conference on Mobile Computing and Networking (MobiCom'95)* (Berkeley, CA, November 1995).

[8] HELMBOLD, D. P., LONG, D. D. E., AND SHERROD, B. A dynamic disk spin-down technique for mobile computing. In *Proceedings of the Second ACM International Conference on Mobile Computing and Networking (MobiCom'96)* (Rye, NY, November 1996).

[9] HUANG, H., PILLAI, P., AND SHIN, K. G. Design and implementation of power-aware virtual memory. In *Proceedings of USENIX Technical Conference* (San Antonio, TX, June 2003).

[10] JOSEPH, A. D., AND KAASHOEK, M. F. Building reliable mobile-aware applications using the Rover toolkit. In *Proceedings of The Second ACM International Conference on Mobile Computing and Networking (MobiCom'96)* (White Plains, NY, November 1996).

[11] KUMAR, R., FARKAS, K., JOUPPI, N., RANGANATHAN, P., AND TULLSEN, D. Single-isa heterogeneous multi-core architectures: The potential for processor power reduction. In *Proceedings of the 36th Annual International Symposium on Microarchitecture* (San Diego, CA, December 2003).

[12] LEBECK, A. R., FAN, X., ZENG, H., AND ELLIS, C. S. Power aware page allocation. In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems* (Cambridge, MA, November 2000).

[13] LITZKOW, M., LIVNY, M., AND MUTKA, M. Condor: A hunter of idle workstations. In *Proceedings of the Eighth International Conference on Distributed Computing Systems* (June 1988).

[14] LORCH, J. A complete picture of the energy consumption of a portable computer. Master's thesis, University of California at Berkeley, December 1995.

[15] LORCH, J. R., AND SMITH, A. J. Reducing processor power consumption by improving processor time management in a single-user operating system. In *Proceedings of the Second ACM International Conference on Mobile Computing and Networking (MobiCom'96)* (Rye, NY, Novemeber 1996).

[16] LUZ, V. D. L., KANDEMIR, M., AND KOLCU, I. Automatic data migration for reducing energy consumption in multi-bank memory systems. In *Proceedings of the 39th conference on Design automation* (New Orleans, LA, June 2002).

[17] MAYO, R., AND RANGANATHAN, P. Energy consumption in mobile devices: Why future systems need requirements-aware energy scale-down. *Lecture Notes in Computer Science* (2003). Special Issue on Power Management.

[18] MUSOLL, E., LANG, T., AND CORTADELLA, L. Exploiting the locality of memory references to reduce the address bus energy. In *Proceedings of the 1997 International Symposium on Low power electronics and design* (Monterey, CA, August 1997).

[19] OLSEN, C. M., AND MORROW, L. A. Multi-processor computer system having low power consumption. In *Proceedings of the Second International Workshop on Power-Aware Computer Systems* (Cambridge, MA, February 2002).

[20] PERING, T., RAGHUNATHAN, V., AND WANT, R. Exploiting radio hierarchies for power-efficient wireless device discovery and connection setup. In *Proceedings of the IEEE International Conference on VLSI Design* (January 2005).

[21] PLANK, J., BECK, M., KINGSLEY, G., AND LI, K. Libckpt: Transparent checkpointing under Unix. In *Proceedings of the USENIX Winter 1995 Technical Conference* (January 1995).

[22] RODRIGUEZ, P., CHAKRAVORTY, R., CHESTERFIELD, J., PRATT, I., AND BANERJEE, S. Mar: A commuter router infrastructure for the mobile internet. In *Proceedings of the Second International Conference on Mobile Systems, Applications, and Services* (Boston, MA, June 2004).

[23] ROLLINS, S., ALMEROTH, K., MILOJICIC, D., AND NAGARAJA, K. Power-aware data management for small devices. In *Proceedings of the Fifth ACM international workshop on Wireless mobile multimedia* (Atlanta, GA, September 2002).

[24] SHIH, E., BAHL, P., AND SINCLAIR, M. J. Wake on Wireless: An event driven energy saving strategy for battery operated devices. In *Proceedings of the Eighth ACM Conference on Mobile Computing and Networking* (Atlanta, GA, September 2002).

[25] SIMUNIC, T., BENINI, L., GLYNN, P., AND MICHELI, G. D. Dynamic power management for portable systems. In *Proceedings of the Sixth ACM International Conference on Mobile Computing and Networking (MobiCom'00)* (Boston, MA, August 2000).

[26] STELLNER, G. CoCheck: Checkpointing and process migration for MPI. In *Proceedings of the Tenth International Parallel Processing Symposium* (April 1996).

[27] THEIMER, M. M., LANTZ, K. A., AND CHERITON, D. R. Preemptable remote execution facilities for the V system. In *Proceedings of the 10th Symposium on Operating Systems Principles (SOSP'85)* (Orcas Island, WA, December 1985).

[28] TIWARI, V., MALIK, S., AND WOLFE, A. Compilation techniques for low energy: An overview. In *Proceedings of the 1994 IEEE Symposium on Low Power Electronics* (October 1994).

[29] WANT, R., PERING, T., DANNEELS, G., KUMAR, M., SUNDAR, M., AND LIGHT, J. The personal server - changing the way we think about ubiquitous computing. In *Proceedings of Ubicomp 2002: 4th International Conference on Ubiquitous Computing* (Goteborg, Sweden, September 2002).

[30] WEISER, M., WELCH, B., DEMERS, A., AND SHENKER, S. Scheduling for reduced CPU energy. In *Proceedings of The First Symposium on Operating Systems Design and Implementation (OSDI'94)* (Monterey, CA, November 1994).

[31] WHITAKER, A., COX, R. S., SHAW, M., AND GRIBBLE, S. D. Constructing services with interposable virtual hardware. In *Proceedings of the First Symposium on Networked Systems Design and Implementation (NSDI '04)* (San Francisco, CA, March 2004).

[32] ZENG, H., ELLIS, C. S., LEBECK, A. R., AND VAHDAT, A. Ecosystem: Managing energy as a first class operating system resource. In *Proceedings of the Tenth international conference on architectural support for programming languages and operating systems* (San Jose, CA, October 2002).

Appendix

The ratio of lifetimes is given by:

$$\frac{L_T}{L_L} = \frac{f_A^L \cdot P_A^L + (1 - f_A^L) \cdot P_S^L}{f_A^P \cdot P_A^P + (1 - f_A^P) \cdot P_S^P + P_S^L}. \tag{7}$$

Using the set of measurements found in Table 3, we find that $P_A^L = 14.8 \cdot P_A^P$, $P_S^L = 0.24 \cdot P_A^P$, and $P_S^P = 0.05 \cdot P_A^P$. Substituting these into equation 7, we find that

$$\frac{L_T}{L_L} = \frac{0.26 + 14.8 \cdot f_A^L}{0.95 \cdot f_A^P + 0.31}. \tag{8}$$

As long as this ratio is greater than 1, Turducken provides a gain in system lifetime. If we define a factor $x = \frac{f_A^P}{f_A^L}$, then the condition becomes:

$$14.8 \cdot f_A^L > 0.95 \cdot x \cdot f_A^L + 0.05 \tag{9}$$

or

$$x < 15.57 - \frac{0.05}{f_A^L}. \tag{10}$$

So, for example if $x = 5$ and the laptop is on for a fraction of time greater than $f_A^L = 0.05/10.57$, or 17 seconds of every hour, Turducken provides a benefit. In other words if the web cache only runs 17 seconds of every hour the StrongARM can be up to 5 times slower at refreshing the cache.

Notes

[1] A Turducken is a Cajun dish made by stuffing a chicken into a duck, which is then stuffed into a turkey.

THE USENIX ASSOCIATION

Since 1975, the USENIX Association has brought together the community of developers, programmers, system administrators, and architects working on the cutting edge of the computing world. USENIX conferences have become the essential meeting grounds for the presentation and discussion of the most advanced information on new developments in all aspects of advanced computing systems. USENIX and its members are dedicated to:

- problem-solving with a practical bias
- fostering innovation and research that works
- communicating rapidly the results of both research and innovation
- providing a neutral forum for the exercise of critical thought and the airing of technical issues

For a complete listing of member benefits, see http://www.usenix.org/membership/bens.html.

Want more information about USENIX? See http://www.usenix.org/ or contact:
USENIX Association, 2560 Ninth Street, Suite 215, Berkeley, CA 94710 USA
Phone: 510-528-8649 Fax: 510-548-5738 Email: *office@usenix.org*

Thanks to USENIX Supporting Members

❖ Addison-Wesley/Prentice Hall PTR ❖ Ajava Systems, Inc. ❖ AMD ❖ Asian Development Bank ❖
❖ Atos Origin BV ❖ Cambridge Computer Services, Inc. ❖ Delmar Learning ❖
❖ DoCoMo Communications Laboratories USA, Inc. ❖ Electronic Frontier Foundation ❖
❖ Hewlett-Packard ❖ IBM ❖ Intel ❖ Interhack MacConnection ❖ The Measurement Factory ❖
❖ Microsoft Research ❖ ❖ Oracle ❖ OSDL ❖ Perfect Order ❖ Portlock Software ❖ Raytheon ❖
❖ Sun Microsystems, Inc. ❖ Taos ❖ Tellme Networks ❖ UUNET Technologies, Inc. ❖ Veritas Software ❖

ACM SIGMOBILE

ACM SIGMOBILE is the Association for Computing Machinery's Special Interest Group on Mobility of Systems, Users, Data, and Computing. Founded in 1947, ACM is the world's first educational and scientific computing society. Today, ACM serves a membership of more than 80,000 computing professionals in more than 100 countries in all areas of industry, academia, and government.

SIGMOBILE is the primary international organization dedicated to addressing the latest developments in all areas of mobile computing and wireless and mobile networking. SIGMOBILE has members from around the world, from academic organizations, industry research and development, government, and other interested individuals. Members of SIGMOBILE are active in the development of new technologies and techniques for mobile and wireless computing and communications.

As a member of SIGMOBILE, you will receiver our quarterly journal, *Mobile Computing and Communications Review* (MC R). You will also be able to receive announcements via our moderated members-only email distribution list, keeping you informed of the latest happenings in our field, including new opportunities to share your research and to meet with friends and colleagues at conferences and other events. SIGMOBILE members also are eligible for the lowest generally available registration fee for any event sponsored or co-sponsored by SIGMOBILE, and for the many events sponsored by other organizations offered in-cooperation with SIGMOBILE.

For more information about ACM SIGMOBILE, visit us on the web at *http://www.sigmobile.org/*. You may also contact the ACM Membership Services Department by email at *acmhelp@acm.org* or by telephone at 1-800-342-6626 (U.S. and Canada) or +1-212-626-0500 (outside U.S.).

ISBN 1-931971-31-5